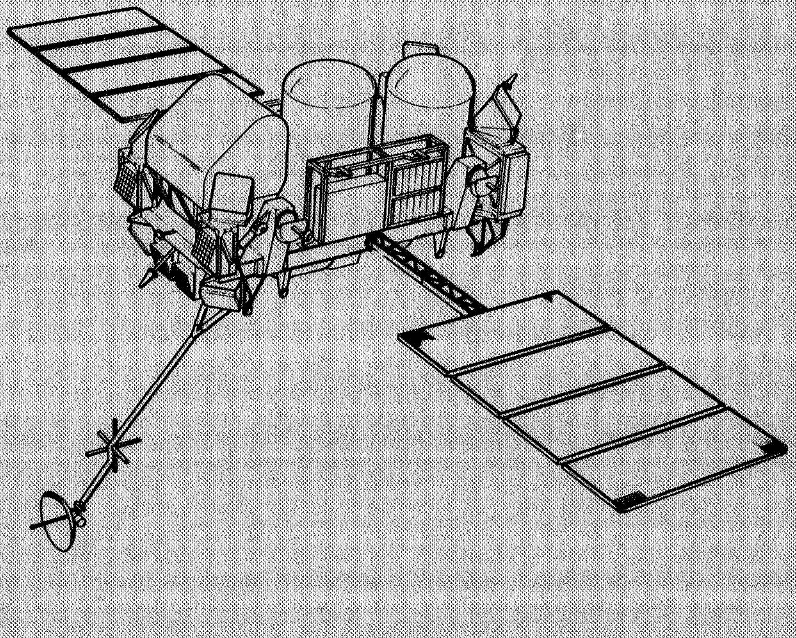
NASA Conference Publication 3110

# 1991 Goddard Conference on Space Applications of Artificial Intelligence



*Proceedings of a workshop held at*
*NASA Goddard Space Flight Center*
*Greenbelt, Maryland*
*May 13–15, 1991*

NASA

*NASA Conference Publication 3110*

# 1991 Goddard Conference on Space Applications of Artificial Intelligence

James L. Rash, *Editor*
*Goddard Space Flight Center*
*Greenbelt, Maryland*

Proceedings of a workshop held at
NASA Goddard Space Flight Center
Greenbelt, Maryland
May 13–15, 1991

National Aeronautics and
Space Administration

Office of Management

Scientific and Technical
Information Division

**1991**

# Foreword

The sixth annual Goddard Conference on Space Applications of Artificial Intelligence is sponsored by the Mission Operations and Data Systems Directorate in cooperation with the American Institute of Aeronautics and Astronautics (AIAA) National Capital Section. The conference provides an opportunity for researchers and practitioners of artificial intelligence in the space industry to gather and share the results of their work. Again this year we anticipate the conference will provide an effective forum for the exchange of ideas, interests, and experiences.

Every year we see the domain of space applications of artificial intelligence expand and the technology mature, but the mainstream operational applications tend to lag behind. This is true of technology transfer in general, and provides us the great challenge of finding ways to facilitate the inclusion of new technology in our systems where it provides meaningful improvements in quality or life cycle cost. Coming together to compare experiences and put forth new ideas will help us meet this challenge.

In the pages that follow are this year's technical papers. They discuss many aspects of the use of artificial intelligence in planning and scheduling and in system monitoring and fault diagnosis. There are also smaller samples of work in neural networks, machine vision, distributed systems, knowledge acquisition and representation, tools, system development, and intelligent tutoring systems. The papers range from theoretical to operational and represent work from universities, private industry, and NASA centers.

Finally, I would like to recognize the many people who have contributed to this conference. Thanks to the authors for providing the technical papers and talks which provide the primary content of the conference. Thanks to the invited speakers and panelists for taking the time to participate and give us their special perspective. And thanks to the conference committee--their dedication and hard work deserve our recognition because they make this event possible, and their participation throughout the year of preparation helped make this an enjoyable experience for me.

Pepper

Jonathan B. Hartley
Chairman
1991 Goddard Conference on Space Applications of Artificial Intelligence

# Table of Contents

***Planning and Scheduling***

***Diagnosis/Monitoring***

### *Robotics/Intelligent Control*

### *Tools & Techniques*

### *Special Topics*

# *Planning and Scheduling*

# OPTIMUM-AIV
# A Planning and Scheduling System for Spacecraft AIV

M. M. Arentoft and J. J. Fuchs, Computer Resources International, Denmark *
Y. Parrod and A. Gasquet, MATRA ESPACE, France
J. Stader, AIAI, University of Edinburgh, Scotland
I. Stokes, Progespace, France
H. Vadon, ESA/ESTEC, The Netherlands

January 31, 1991

## Abstract

This paper presents a project undertaken for the European Space Agency (ESA). The project is developing a knowledge based software system for planning and scheduling of activities for spacecraft assembly, integration and verification (AIV). The system extends into the monitoring of plan execution and the plan repair phases.

The objectives of the contract are to develop an operational kernel of a planning, scheduling and plan repair tool, called OPTIMUM-AIV, and to provide facilities which will allow individual projects to customize the kernel to suit its specific needs. The kernel shall consist of a set of software functionalities for assistance in initial specification of the AIV plan, in verification and generation of valid plans and schedules for the AIV activities, and in interactive monitoring and execution problem recovery for the detailed AIV plans. Embedded in OPTIMUM-AIV are external interfaces which allow integration with alternative scheduling systems and project databases.

The current status of the OPTIMUM-AIV project, as of January '91, is that a further analysis of the AIV domain has taken place through interviews with satellite AIV experts, cf. [6], a software requirements document (SRD, [7]) for the full operational tool has been approved, and an architectural design document (ADD, [8]) for the kernel excluding external interfaces are ready for review. At the time of the conference the implementation will be well underway expecting a final delivery in September of '91.

## 1 Introduction

The size and complexity of the tasks involved in the AIV of spacecraft, raises the need for efficient and flexible planning and scheduling tools. An evaluation of the current available and applied commercial tools reveals their inadequacies towards the general problem of AIV.

In 1988 this lead ESA to award a contract to a consortium consisting of CRI, MATRA ESPACE and AIAI, which should assess the applicability of AI and KBS techniques in a prototype AIV planning and scheduling tool. This study resulted in a set of user and software requirements and a demonstration system exploring some of the aspects of AIV planning, cf. [5].

OPTIMUM-AIV is a follow-up project carried out by CRI, MATRA ESPACE, AIAI, and Progespace. The objectives of the project are three-fold:

1. to develop an operational kernel of a planning, scheduling and plan repair tool consisting of a set of software functionalities for assistance in
   - initial specification of the AIV plan
   - generation of valid plans and schedules for the various AIV activities
   - interactive monitoring of the AIV plan execution
   - identification of immediate effects and plan repair of problems
2. to embed external interfaces which allow integration with alternative scheduling systems and project databases.

*Bregnerødvej 144, DK-3460 Birkerød, Denmark, Phone: + 45 4582 2100, Fax: +45 4582 1766, E-mail: mma@spd.cri.dk

3. to provide facilities which will allow individual projects to customize the kernel to suit its specific needs

The realization of these objectives are explained in the sections to come.

The outline of the paper is as follows. First section 2 outlines the operations domain of spacecraft AIV planning, and the benefits and applicability of OPTIMUM-AIV to this domain are introduced in section 3. Based on this outline section 4 lists the explicit domain dependent knowledge to be included in the tool. Before the detailed discussions of the main tool components section 5 provides an overview of the system process stages. Next section 6 shows how plan specification admits the user to consult libraries of past and generic plans and section 7 explains how the generation of plans takes into consideration the logical precedence ordering between activities and specifications of the expected outcome and required configuration of the spacecraft equipment being put together and tested. Then the satisfaction of temporal and resource usage constraints are described. Afterward, the execution monitoring and plan repair is presented, in section 8, as plan status updating, progress interpretation, and consistency checking and recovery organized along identified execution problems. Subsequently, section 9 lists the external interfaces to be embedded in the system and explains about their intended use. Finally, the lessons learned wrt. use of AI techniques in the system are discussed, in section 10.

# 2 Operations Domain: Spacecraft AIV Planning

This section gives a brief outline of the AIV planning process and life cycle, and hence establishes the function and purpose, environmental considerations, and general constraints of OPTIMUM-AIV.

Spacecraft development projects are typically divided into the following phases:

**A** : Early feasibility study:
The overall mission objectives of the intended programme are evaluated and a feasibility assessment is made based on operational constraints. This serves as a basis for deciding whether the project should be undertaken or not. The goals of the phase are to derive system requirements, to establish a preliminary model philosophy, and to identify verification aspects and assess their influence on the spacecraft design. An early identification of the needed verification tools and a general planning of the programme and the AIV aspect are also undertaken.

**B** : Specification phase:
System requirements are extracted in a top-down manner, starting with the total system and ending up with specifications of the various primitive units. The phase is critical to AIV since it must clearly define the AIV approach to be taken in phase C/D. During phase B the general AIV plan, additional facilities plans, ground support equipment (GSE) requirements, test hardware requirements and development plans at lower levels are produced. The general AIV plan is part of the overall project management plan.

**C/D** : Development and integration phase:
In this phase the design is frozen and manufacturing is undertaken. This is mainly a bottom-up activity where primitive units are put together to form assemblies, assemblies to subsystems, and subsystems are integrated at the system level. The phase is completed with the integration, verification and qualification of the spacecraft system.

This phase implements the plans generated in phase B and produces detailed AIV plans at different levels. This involves propagation of logical constraints, assignment of dates to activities logical flow, verification of resource consumptions vs. availability, and monitoring of the execution, i.e. updating of activity statuses and handling of failures.

**E** : Operational phase:
This last phase covers the period from the launch until the end of the spacecraft mission. Verification as such is completed before this phase. The electrical GSE has served its purpose and the satellite is instead controlled and operated by the ground segment.

The phases described above defines the AIV life cycle:

**Elaboration** : Phases A+B:
Philosophy and model(s) are selected, e.g. prototype, protoflight, or project specific philosophy, and structural, thermal, electrical, protoflight, or flight model. Furthermore an evaluation of project parameters takes place, for instance of due dates, manpower, GSE, and cost aspects.

**Implementation** : Phase C/D:
Detailed AIV plans, at different levels and with various time windows, are generated. The required initial configurations and effects of activities are compiled into a sequencing logic, and time and resources are verified and assigned. The resulting schedules are documented in detailed and summary networks.

**Monitoring** : Phase C/D:
The sequencing logic, temporal and resource usage constraints, and possibly AIV objectives, are reviewed in case of disturbance. The impacts are analyzed and critical and degraded activities are identified. New AIV plans are produced taking into account the current constraints and additional tasks for repair.

Phase E is only relevant if there are AIV activities during the operation of the spacecraft, e.g. if a reusable module has to be integrated again.

## 3 System Objectives

This section introduces the benefits and applicability of OPTIMUM-AIV.

The planning tool will assist primarily at the AIV team leader level in the management of day-by-day activities, and secondarily at the project group (interface) level.

The tool will cover the phases B and C/D. In phase B project management will define the high level AIV plans, which must be refined and detailed in phase C/D in order to constitute operational plans. OPTIMUM-AIV will provide a dynamic environment in which the AIV plans can be input and refined using AI based planning and scheduling techniques. Also in phase C/D, the constructed plans are executed. OPTIMUM-AIV will be used to monitor the progress, and to assist the users in identifying and solving any unforeseen problems and in producing new plans.

OPTIMUM-AIV shall provide the following facilities:

- definition, from scratch and as extension, of the AIV plan
- derivation and construction of plans and schedules at several levels
- monitoring and assistance in replanning of project execution

Especially on the last point the planning tool will differ from current planning systems, and assistance in replanning will indeed be the principal objective of OPTIMUM-AIV. The advantage of using knowledge based techniques will be a more flexible system in which planning knowledge is explicitly represented. The knowledge concerning conflict resolution and replanning will be incorporated and used to assist the users in generating feasible plans and solving problems during plan execution.

## 4 System Knowledge

One important aspect of OPTIMUM-AIV, and an aspect which makes it different from traditional planning and scheduling tools, is the inclusion of explicit domain knowledge.

We distinguish entity knowledge from process knowledge.

**Entity knowledge** defines and represents the entities that must be manipulated in the domain. For the AIV planning domain the entity knowledge is the knowledge concerning spacecraft systems and models, generic, past and current projects and plans, AIV activities, resources, and global constraints.

**Process knowledge** represents the knowledge stating how the entity knowledge manipulation may be done. For the AIV planning domain the process knowledge is the general planning and scheduling knowledge, plus explicit heuristics and knowledge about the rationale behind the plan structure.

The entity knowledge is represented as objects in classification hierarchies, which are also applied in the current management of large AIV programmes.

Spacecraft systems and models are decomposed in a part-of hierarchy.

Projects and plans are modularized so that each subplan may be worked on independently without necessarily having to load the full ensemble of AIV plans.

AIV activities have been classified according to their function in the AIV process, e.g. reception, preparation, assembly, integration, functional/performance test, environmental test, etc. Alternative classification dimensions could be the technological nature of activities or the spacecraft system which is the object of the activity. Indeed the AIV activity hierarchy may be mapped onto the spacecraft part-of hierarchy through relations expressing that an AIV activity is performed on certain spacecraft subsystems. The use of this type of mapping is to verify the actual AIV plan against rules and policies for subsystems in the spacecraft decomposition.

The description of an activity carries over to the description of a project, or plan, since a plan is simply a description of a larger capability, or macro-activity. This macro-activity is expanded into a subplan of child activities, which in turn may be expanded into more detailed subplans, and so on. Each subplan is independent from other subplans in the sense that it has a unique starting point and a unique completion point which are exactly equal to the start and completion of the parent activity, i.e. each activity is decomposed into sub-activities independently.

Individual activity descriptions contain information about:

- preconditions which must hold for the activity to be appropriate and to be triggered,
  e.g. house.keeping.module CONNECTED.TO
  data.handling.subassembly
- effects (and side effects) of using the activity,
  e.g. tm-tc-loc INTEGRATED.IN
  house.keeping.module
- constraints, including time and resources, e.g. the activity requires two electricians for three days
- the activity itself such as its objectives, documentation, etc.

An activity is generally regarded as a plan fragment in its own right. Hence there may be partially ordered activities contained within the description. However the planning system assumes responsibility for completing the partial order description of the final plan. These descriptions are also generally parameterised, generic descriptions which are instantiated at the time of use. This offers flexibility and assists with a least commitment approach to planning and scheduling.

Resources have been categorized along two dimensions. First, according to predefined resource classes, e.g. GSE, manpower, test facilities, money, etc. Secondly, according to the nature of the resource, i.e. shared or consumable. Resources are shared if their availability must be specified as a function of time, e.g. manpower. Resources are consumable if there is an initial stockpile available which can only be depleted by activities in the plan, e.g. money.

Resource descriptions contain information about:

- availability profile, e.g. the resource is present during the third week of May
- alternative and indirect resources
- the resource itself

As in the case with spacecraft systems and activities the resources are also classified in an object hierarchy where generic functions are inherited to the specific resource instances.

Activity global constraints can be associated with a schedule. They express overall temporal relations between activities in a certain context. The context is defined by a given status of the involved activities and a certain configuration of the spacecraft system. The context may contain arbitrary variables that may be related in general predicates. For instance we have:

```
IF     ACTIVITY acoustic.and.vibration.test
       of.Class environment.test
       works.on.System ?s
       ∧
       ACTIVITY ?x
       of.Class integration
       works.on.System ?e
       ∧
       SYSTEM ?e SUBELEMENT.OF ?s
       ∧
       ?x ≠ power.supply.subsystem.integration
THEN   acoustic.and.vibration.test AFTER ?x
```

The process knowledge is represented as rules and tables recording user preferences and decisions.

Heuristics are associated with projects and/or individual activities. Heuristics are different from constraints; they are used to decide on strategies in order to restrict the search space. These strategies are applied when all constraints have been satisfied and the system is left with degrees of freedom allowing it to follow user preferences.

The rationale behind the plan structure is written in a structure that explains how certain spacecraft system configurations have been satisfied at specific points in the current plan and records the alternative satisfaction means that were considered. This structure is generated during planning and scheduling and used to restore consistency when execution problems occur. The idea of rationale recording originates from [2], [3], and [4].

## 5 System Process Stages

An explicit distinction of the different stages of the system processes helps clarify the purpose and rationale of the system functions:

- Knowledge Editing and Plan Specification
  - Definition and input of general domain knowledge: spacecraft system, activity, resource, global constraint classes and instances
  - Specification of the actual planning problem: project events and strategies, relations between domain object instances
- Plan and Schedule Generation
  - Planning: find a logically valid plan in sufficient detail
  - Scheduling: include time and resources in the plan
- Project Monitoring and Plan Repair
  - Monitoring the execution of the schedule: record the progress, remind the user of activities to be started soon
  - Detection of problems and their immediate impacts: derive local inconsistencies
  - Schedule repair: reschedule or edit the current schedule locally, e.g. up to the next milestone
  - Plan repair: more serious problems may interfere with the plan logic

The first stage requires mostly user input and thus editing facilities, and some support like input validation etc. The latter stages make use of the information gained at this first stage and they may require additional information from the user. These latter stages are more interactive. In particular the monitoring and plan repair stage require an extensive dialogue with the user.

The second stage build the plan and the schedule in advance of it being required. It also records justifications for planning decisions taken by the user in resolving conflicts.

The predictive approach adopted for the second stage is poor on recovery when failure arises. Since the third stage must support ease of repair we have chosen a more reactive approach for the project monitoring and plan repair stage. It will proceed forward in time only, using dependency recording techniques aimed at enabling repair to be limited to only those components known to be affected by the forced changes.

The different approaches for plan and schedule generation and for execution problem recovery reflect the fact that

> "... there is a trade-off between the predictability of the environment in which the plan is to be executed and the degree of reactivity which is necessary to successfully achieve the goals of the plan ...",

quoting directly from [1], p. 124. OPTIMUM-AIV is capable of admitting both a predictive approach and ease of repair.

There is not an overall strategy management set of rules, as it is doubtful if the human AIV expert will be able to balance the requirements of the technical objectives along with each of the strategic objectives. Rather the system recognises the complexity of objectives and devises a multi-perspective strategy aimed at meeting *all* of the objectives, rather than a single, or minimal, perspective focussed around some of the objectives such as handling all AIV tasks, meeting all deadlines, or matching resource demands with availabilities, e.g. estimated costs to budgeted costs.

## 6 Knowledge Editing and Plan Specification

In this initial stage most of the work relies on the user inputting and specifying the AIV activities, their required spacecraft components, resources, and interfaces, as well as their decomposition which will eventually constitute a plan for the project. This information appears from the AIV plan defined by project management in phases A and B of the spacecraft AIV life cycle. The information is entered through structured editor environments that has a simple compiler convert the input to internal form.

The system enables the user to retrieve information from past AIV plans and to browse into the activities of these plans. The past plans are indexed according to their main characteristics: the type of the spacecraft system, the types of AIV models used, and the subsystem of the spacecraft for which the plan has been applied. It is thus possible to incorporate heuristics specifying typical scenarios and durations of certain activities and to assist in the assessment of project duration, resource consumption and cost. In this way optimistic, probable, and pessimistic estimates can be based on experience.

The experience has been recorded during past AIV programme executions. The recording is facilitated through a node pad facility and special activity, resource, and global constraint attributes, where the user may record previous results of using the system. Here the user may comment on the actual performance of an activity, his experience in using a resource or in applying a global constraint. These experience attributes complements the information that can be derived from comparison of estimated versus planned values.

The case-based approach of using past plans is combined with the use of generic plans. Generic plans specify a number of typical activities for a certain (component of a) spacecraft system. The assumption is that general principles of spacecraft AIV may guide the initial plan establishment. The activity attributes could define in which order they must be undertaken. Generic plans, or prototype plans, are thus a collection of imaginary AIV activities which must typically be undertaken to perform the ideal AIV process for a selected ideal system or model. They are generic in the sense that no actual programmes will ever be able to use the plans without making modifications to them. Furthermore they are generic in the sense that all the generic activities and the generic resources must be instantiated to represent the actual world. That is, scheduling information, precise resource specifications, etc. must be added in order to properly instantiate the activities to represent an actual plan, or schedule. OPTIMUM-AIV provides mechanisms which allows the user to search through the various generic plans and to make instantiated versions which can be used as the basis for actual AIV plans.

# 7 Plan and Schedule Generation

## Constraint Satisfaction

During the plan and schedule generation stage we distinguish five kinds of activity constraints. These constraints are imposed by the environment of the planning problem, e.g. resource usage constraints, or by the nature of the problem, e.g. precedence constraints.

**Precedence :**
These predecessor and successor constraints specify explicit user defined orderings on the activities. The constraints are expressed as directed links between activities They may be used to constrain the temporal specifications of the activity. That is, if any of the predecessors or successors are committed to certain time feasibility windows, then this may reduce the duration of the possible time window of the current activity.

**Precondition :**
These are the more general constraints, which may specify that certain results must have been obtained, or some equipment be available before the activity can be undertaken. The former type of condition may be used to constrain the plan network by adding predecessor links to the activities which have the required condition as an effect. The latter type of constraint may be used to expand the current plan through the addition of e.g. transport activities which will have the required condition as its effect.

**Temporal :**
Time is one of the major constraining factors in the spacecraft AIV planning domain. Temporal constraints are manifested in a number of ways. First, through the specification of delivery and completion dates, which the various activities must be scheduled to satisfy. Secondly, as a maximum duration which the activity must be undertaken within. These types are what may be called absolute temporal constraints. Furthermore a number of relative or second-order temporal constraints may be deduced. They are deduced on the basis of precedence relations and on the possible temporal limitations on the availability of e.g. resources. Indeed the establishment of temporal specification from precedence relations

is one of the major activities in traditional OR scheduling algorithms.

**Resource Usage :**
This type of activity constraint specifies which and how much of various resources an activity demands. The constraints are expressed as references to resource instances, and a specification of the required consumption profile. The information is used to constrain the time feasibility windows in which the activities can be scheduled.

**Global Activity Constraints :**
Global activity constraints express overall temporal relations that must hold given a status of the involved activities and of the spacecraft system.

Verification of precedence and precondition constraints takes place during planning, temporal and resource usage constraints are propagated during scheduling, whereas global activity constraints can be satisfied at any system process stage. Typically, however, global activity constraints will be checked after scheduling by simple goal processing, or backward chaining, through the context/temporal relation rules.

## Planning

Plan generation entails verification of the plan logic, assistance in conflict resolution, and construction of new precedence relations based on preconditions and effects of activities. The basis is the initially specified AIV plan which must be refined and detailed.

The plan logic verification is divided into checking of user-defined precedence relations between activities and validation of preconditions and effects of activities vs. actual spacecraft system states.

The checking of precedence relations includes detection and resolution of dangling references to predecessor and successor activities, and of cycles specified by the precedence links.

The validation of activity and spacecraft system states checks for interactions between parallel activities and propagates system configurations from the start activity to the project due date. The propagation ensures that the effects of one activity do not violate the preconditions of a succeeding activity, i.e. that the ordering of activities is consistent with the preconditions and effects specified for each activity. There might be two types of conflicts: a precondition of an activity is

1. in conflict with the actual state of the system
2. not found in the actual state of the system

Possible modifications to restore the consistency of the plan logic are

- modification of the preconditions and/or effects of one or many activities
- change in the precedence relations between activities
- addition or deletion of activities to introduce or avoid the configuration in question

## Scheduling

Schedule generation involves management of temporal and resource usage constraints; verification, conflict resolution, and construction. The relation between an activity and its decomposition is an active relation in the sense that definitions and changes made at one level propagate to the other levels in the plan hierarchy. This holds true both for time definitions and resource assignments.

Time feasibility windows (TFW) for the activities are calculated by a forward and a backward pass of the logical plan. These passes work on estimated bounds, rather than exact values, as specified by the user for activity durations and times.

Within the TFW's the system places the activities according to the local strategy associated with each activity. If a local strategy is not specified the project as a whole has defined a global strategy which determines the preferable assignment of actual times to activities.

These preferences regarding activity placements may be overruled by violation of resource constraints. The management of shared resources, called resource smoothing, is an inherently intractable problem. We have constructed the following simple heuristics for resource smoothing: calculate the shared resource usage profile, compare this profile with the availability profile, shift activities within their TFW's if necessary, and, if shifting is not sufficient, solve the overconsumption problem in cooperation with the user.

The management of consumable resources is simplified by the fact that the timing of their usage is unimportant. Only the total usage is relevant and the system must simply guarantee that this does not exceed the initial availability. Our problem is one of representing and propagating resource consumption constraints in an efficient manner.

By specifying upper and lower bounds on resource usage it is a relatively simple and flexible task to track the more detailed specification of actual resource usage as a plan is refined into lower levels of detail, as in hierarchic planning. In fact the maintenance of bounds makes this a useful checking and pruning mechanism, as the expectation should be that lower levels of detail merely provide a more accurate specification of actual use and should not be unnecessarily constrained by complete specifications at higher levels.

Although the resource management algorithms bear some similarity to the maintenance of temporal constraints there are important differences which make resource management of a strictly consumable resource easier, cf. [4], p. 21:

- time is not a resource because the consumption of time by activities in parallel is independent of the number of actions in progress. Thus it is inconvenient to represent the consumption of time by a particular activity.
- the sheer number of temporal constraints is often far greater than the number of resource constraints.
- Resource constraint propagation is the maintenance of a conjunction of constraints of a particularly simple form. With temporal constraints, the planner must often impose additional constraints to satisfy a condition. Such constraints are often a disjunction of simpler constraints.

### Knowledge About Plans and Schedules

During the planning and scheduling processes information is created about the proposed plan. This information consists not only of the plan structure itself but also of information about the structure. Together these make up a knowledge rich representation of the plan. Information about the reasons for ordering activities in a certain way, about which states and domain objects are affected by which parts of the plan, and about which activities and/or links were introduced into the plan for what purpose is recorded at this stage. Individual conditions required in a plan are deliberately tied to the effects which will necessarily ensure their satisfaction.

The advantage of a knowledge rich representation is that it enables the system to reason about the plan. This is a necessary requirement for determining immediate impacts of changes to the plan during execution monitoring and for assistance in consistency recovery and plan repair. It is also a useful precondition for detecting and possibly avoiding conflict situations and for explaining and justifying planning decisions.

## 8 Project Monitoring and Plan Repair

The major use of OPTIMUM-AIV will be during the plan execution phase. This phase covers the period from the time when the AIV plan starts to be executed until the planned process is completed.

In any planning domain there is a possibility that problems occur when the plan is executed. We distinguish between usual plan failures and plan failures which are specific to the AIV domain: the failure of tests.

The usual plan failures are caused by changes in the actual environment in which the plan is executed. They are often caused by unexpected events or organisational issues like unavailable resources or supplies. When problems occur during plan execution the original plan becomes, at least partly, invalid and it has to be revised. This can be done by replanning where the original plan is discarded and a completely new plan is generated that takes into account the state of execution and the changes of circumstances that lead to the plan failure. In domains like AIV where activities are heavily interdependent to external activities or external resources this approach is not acceptable. Based on the original plan, bookings have been made and temporal interfaces have been specified. It is important that these interfaces are changed as little as possible and thus as much as possible of the original plan is to be retained. In these cases the plan has to be repaired.

The failure of tests make the original plan invalid just like the other problems mentioned above. However, they are expected as possible outcomes of tests and thus in most cases lines of action are predefined, as part of the domain knowledge, to deal with the situation of a test failure. There are various strategies that can be adopted depending on the type of failure. These strategies, or problem scenario subplans, are represented as special cases of generic plans. Some problem scenario plans may be included in the system from scratch, but most will be entered during the actual use of the system in the plan execution phase.

The severity of plan failures varies. Some plan failures have very local effects and problems can be solved by local schedule repair. It may suffice to move start and finish times of a few activities, or to make use of alternative resources or non-nominal availability profiles. However, sometimes this local change of the schedule has effects on other parts of the schedule, e.g. when the alternative resource is scheduled for another activity. In these cases more global changes are necessary to generate a new valid schedule. Effects of plan failures can become so serious that the logic of the plan is affected. It may be impossible to schedule still outstanding activities using the original plan under the given conditions. Then it becomes necessary to repair the actual plan itself rather than just the schedule in order to retain consistency.

The issue of plan repair and schedule repair is very much a research issue in AI planning and thus a fully automated solution to these problems is far beyond the scope here. OPTIMUM-AIV *assists* the user in schedule and plan repair in an interactive way, rather than *performing* repair itself.

The possibility of plan failures, whether expected or not, and the need for plan repair require the monitoring of plan execution. In the simplest case the changes in the current state of execution are given when a plan failure occurs. However, a plan which has not been updated for a long period while work has been undertaken will be more difficult to recover. In the AIV domain where plan failures are frequent and there is need for fast plan repair it is necessary to monitor plan execution more closely. It is important that the plan is kept up-to-date regularly without much effort. Therefore the system gives strong facilities for entering the execution progress and for making small and large adjustments to the plan.

The system gathers information about the actual progress to have basis for determining whether the plan has failed. Then it uses that to determine how execution goes, i.e. to detect discrepancies between the proposed schedule and the monitoring information. This in turn is used to identify what parts of the schedule are inconsistent with, or affected by, the execution state. The system assists in changing the schedule by displaying relevant information in trying to repair the schedule. Finally it is checked whether the new schedule, user or system generated, is consistent.

The consistency checking and recovery is organized along execution problem types caused by user changes. We assume that the user is quite capable of solving execution problems him/herself and will use the system to speed up the process, to make sure that nothing is forgotten and to explore different solutions, what-if scenarios. For each execution problem type there are specific standard recovery methods that are most appropriate for solving the problems. However most problems can be solved in other ways too, which will also be available for use.

# 9 External Interfaces

## ARTEMIS Interface

The system will have embedded an interface to the widely spread ARTEMIS scheduling tool. The interface is intended to be used primarily for

- import of space project data, i.e. activities and events (but not interfaces and hammocks), constraints, and resources datasets,
- export and display of plans, and
- report writing and graphics
- aggregation, i.e. summarize numeric data held in network datasets, e.g. resource requirements for all activities.

It can also be used for network construction, examination of the network logic, time analysis and updating, resource-limited or time-limited scheduling, and multiple network processing. However, in these latter uses of ARTEMIS it will not be feasible to return the results directly to OPTIMUM-AIV.

## Database Interface

OPTIMUM-AIV will be able to interface to satellite related project databases. This reduces the work to input data into the system, and ensures the coherence between external satellite databases and the system database.

It allows the loading of activities, resources and constraints from such external databases. The management system and the format of those databases might vary but they are convertible into relational tables. Therefore the system essentially provides an SQL interface to fill its internal database.

## Programming Interface

The system is designed such as to allow external documentation programs to be written. It provides an interface that permits any user to develop their own documentation, in particular any new representation of the plan and schedule. That means that all activities, resources and constraints and any schedule will be accessible by any external program (written in C, Pascal or Ada).

## 10 Applied AI techniques

This section extracts AI planning and scheduling techniques integrated in the system. The applied techniques complement existing features of current project management tools, and in particular of ARTEMIS.

OPTIMUM-AIV adopts the *non-linear planning* paradigm which enable plan representation to contain causally independent activities which can be executed concurrently. It searches through a space of partial plans, modifying them until a valid plan/schedule is found.

Another important characteristic of the system is *hierarchical planning*. The term hierarchical refer to both the representation of the plan at different levels, and also the control of the planning process at progressively more detailed levels.

The scheduling task is considered as a *constraint satisfaction* problem solved by constraint-based reasoning. The constraints are propagated throughout the plan, gradually transforming it into a realizable schedule. Invariably not all of the constraints can be met, such that some have to be relaxed.

During plan specification and generation the system operates on *explicit preconditions and effects of activities* that specify the applicability and purpose of the activity within the plan. With this knowledge it is possible to check whether the current structure of the plan introduces any conflicts between actual spacecraft system states, computed by the system, and activity preconditions, which have been specified by the user. Such conflicts would arise if one activity deletes the effect of another thus removing its contribution to the success of a further activity. The facility for checking the consistency of the plan logic, by dependency recording, is not possible within existing project management tools, that assumes that the user must get this right.

Also during planning, the system *records the rationale* behind the plan structure i.e. user decisions on alternatives are registered. This is used to assist during plan repair where the user tries to restore consistency. Information can then be derived about alternative activities, soft constraints that may be relaxed, and potential activities that may be performed in advance.

## 11 Conclusion

The current status of the OPTIMUM-AIV project, as of January '91, is that a further analysis of the AIV domain has taken place through interviews with satellite AIV experts, cf. [6], a software requirements document (SRD, [7]) for the full operational tool has been approved, and an architectural design document (ADD, [8]) for the kernel excluding external interfaces are ready for review. At the time of the conference the implementation will be well underway expecting a final delivery in September of '91. It is foreseen that the implementation will be in Common Lisp and Common Lisp Object System.

The domain analysis has identified areas of AIV expert knowledge which must be incorporated into the system. The SRD has derived a model of the domain and has established the functional and operational requirements which must be satisfied to meet the demands of the AIV experts. The ADD has proposed a design which supports the typical mode of interaction between the user and the system. The AIV experts input the plan specifications and revisions and the system goes through a series of analyses, and identifies, visualises, and assists the user in case of conflicts.

The paper has discussed in detail the requirements of the AIV experts, the obtained results with regard to knowledge elicitation and requirement formalization, and the results of the design phase in terms of constraint identification and satisfaction and execution problem detection and recovery.

## References

[1] W. Swartout. *DARPA Santa Cruz Workshop on Planning.* 1987.

[2] A. Tate. *Goal Structure: Capturing the Intent of Plans.* European Conference on AI, Pisa, Italy, September 1984.

[3] D.E. Wilkins *Domain Independent Planning: Representation and Plan Generation.* Artificial Intelligence, 22, 1984.

[4] K. Currie and A. Tate. *O-Plan: the Open Planning Framework.* Final report to the SERC on Alvey grant number IKBS-151, AIAI, 1989.

[5] J.J. Fuchs, G.S. Pedersen, and A. Gasquet. *EPS-AIT: Final Report.* EPS-AIT-CRI-FR-0001-89, CRI, MATRA, AIAI, 1989.

[6] Y. Parrod and I. Stokes. *OPTIMUM-AIV: AIV Knowledge Acquired.* OPT-MATRA-TN1000-0890, MATRA, Progespace, CRI, AIAI, 1990.

[7] M.M. Arentoft, Y. Parrod, and J. Stader. *OPTIMUM-AIV: Software Requirements Document.* OPT-CRI-SRD-1090, CRI, MATRA, AIAI, Progespace, 1990.

[8] M.M. Arentoft, Y. Parrod, J. Stader, and I. Stokes. *OPTIMUM-AIV: Architectural Design Document.* OPT-CRI-ADD-0291, CRI, MATRA, AIAI, Progespace, 1991.

N91-22771

# TDRSS Momentum Unload Planning

George R. Cross
Mitchell A. Potter
J. Douglass Whitehead
James T. Smith

Intelligent Systems Laboratory
Contel Technology Center
15000 Conference Center Drive
Chantilly, VA 22021-3808

**Abstract**

We describe a knowledge-based system which monitors TDRSS telemetry for problems in the momentum unload procedure. The system displays TDRSS telemetry and commands in real time via X-WINDOWS. The system constructs a momentum unload plan which agrees with the preferences of the attitude control specialists and the momentum growth characteristics of the individual spacecraft. During the execution of the plan, the system monitors the progress of the procedure and watches for unexpected problems.

## 1 Introduction

This paper describes a knowledge-based system to assist in the operation of the White Sands Ground Terminal (WSGT) for the Tracking, Data and Relay Satellite System (TDRSS). We first review knowledge-based systems that have been developed in support of spacecraft operations. We also review the specific systems previously developed in the context of TDRSS operations. Since our system acts as an assistant in the planning and execution of TDRSS momentum unloads, we present some background material on the concepts of momentum unloading in body-centered spacecraft. We then describe the problem and tool selection for the momentum unload system and give the current status of the system.

## 2 Expert Systems in Spacecraft Operations

### 2.1 Overview

The applications of expert systems to space systems operations can be broken into three main areas: planning/scheduling, diagnostics, and operations monitoring. We describe example systems in each of these areas.

*Planning Systems* – Planning and scheduling systems determine times when on-orbit assets can be used without conflict and may operate in real-time or more reflectively off-line. For example, Miller [10] reviews expert systems for the Hubble Space Telescope. The obvious problem is to determine and optimize the sequence of observations but there is also an off-line system called the Proposal Entry Processor [5] which examines proposals for the use of the telescope and searches for duplication among proposed observations.

*Diagnostic Systems* – The key issue in on-orbit spacecraft diagnosis and repair is that failing components often cannot be repaired. The few repairs to spacecraft by the Space Shuttle have been accomplished only for those in low earth orbit, not geosynchronous orbits. As a result, repair of satellites in geosynchronous orbit is usually limited to swapping in redundant components. Keller *et al.* [6] describe the development of a rule-based expert system directly from a qualitative and quantitative description of the Reaction Wheel Assembly of the Hubble Space Telescope which appears useful in the pre-launch checkout phase. Since spacecraft are so inherently complex, automatic generation of diagnostic systems is necessary from model-based descriptions if there is any hope of breaking the knowledge acquisition bottleneck. The involvement of experts is still needed to refine system characteristics.

*Spacecraft Monitoring Systems* – Monitoring on-orbit spacecraft requires responsive ground software and a flexible user interface to aid the operational personnel in making decisions. Recent examples of such systems include the Real-Time Data System [11] which is being integrated into mission control operations for space shuttle missions and the Spacecraft Health and Reasoning Prototype (SHARP) [7] which is being used for monitoring deep space probes like Voyager.

## 2.2 TDRSS Expert Systems

Previously constructed expert systems for TDRSS can be split into two general categories: those concerned with on-orbit problems and those concerned with problems in the communications network. Tang and Wetzel [15] discuss a diagnostic system for the NASA Ground Terminal. The equipment in the ground terminal is concerned with data transport, data quality monitoring and line outages. FIESTA [8] diagnoses the end-to-end TDRSS network from WSGT over to the actual site where data is presented to the end user. On-orbit TDRSS systems include ESSOC [13] which assists operators performing a delta-v maneuver on TDRSS Flight 1. ESSOC could use live TDRSS telemetry but was not deployed. MOORE [4] and its successor PACES [1] deal with the diagnosis of problems in the attitude control subsystem hardware but are primarily demonstration programs not intended for operational use.

# 3 TDRSS Operations

## 3.1 Momentum Unload Planning

### 3.1.1 Yaw Momentum

During nominal TDRSS operations [3, Volume 1, Book 1, 3-115ff] there is an angular momentum which is *normal* to the orbit plane. The angular momentum vector, through solar radiation pressure torques, picks up a component *in* the orbit plane. This is called the *transverse momentum vector* and is denoted by $H_t$. We can resolve $H_t$ as two components $H_{tx}$ and $H_{tz}$. $H_{tx}$ lies along the projection of the spacecraft roll axis into the orbit plane while $H_{tz}$ lies along the projection of the spacecraft yaw axis onto the orbit plane. When there are no attitude errors, these are both exactly aligned with the roll and yaw axes. The *yaw attitude error* or *yaw angle* is related to $H_{tx}$ by

$$\psi = \arctan(\frac{-H_{tx}}{H_n}) \tag{1}$$

where $H_n$ is the instantaneous angular momentum along the orbit normal and has a nominal value of -32.535 ft-lbs-sec ($fps$).

The other component $H_{tz}$ is expressible in terms of the reaction wheel speeds $\omega_1$ and $\omega_2$ by the relation in Equation 2 below. The number 366.8 is a conversion between the reaction wheel speed differences and torque expressed in $fps$ [3, Volumel, Bookl, p3-39]

$$\omega_1 - \omega_2 = 366.8 \times H_{tz}. \tag{2}$$

### 3.1.2 Purpose of Momentum Unloading

The term *secular torques* means the torques caused by slowly-varying phenomena, principally solar disturbances. Momentum unloading is required to reduce the effects of secular torques. In particular, yaw momentum unloading controls the quantities defined in Equations 1 and 2.

The first constraint is that $\psi$ remains under 1.0 degrees (in radians, $\pi/180$). To see the effect of this, we first assume that $|\psi| \leq \pi/180$. After making a substitution for $\psi$ from Equation 1, applying the tangent function to both sides, and accounting for the possibility of both positive and negative deviations, one obtains the relation

$$|H_{tx}| \leq \tan(\pi/180) \times H_n = 0.01745506 \times H_n. \tag{3}$$

More simply, assuming that $H_n$ has its nominal value of -32.535 $fps$,

$$|H_{tx}| \leq 0.56790054 \approx 0.57 fps. \tag{4}$$

It also is required that the maximum yaw momentum is below the level that can be controlled by the roll/yaw integrator. The maximum that can be controlled is $1.50fps$, and a margin of error of $0.5fps$ is assumed. Thus, the maximum for $H_{tz}$ is set at $1.0fps$. Setting $|H_{tz}| \leq 1.0fps$ and using Equation 2 yields the result

$$\omega_1 - \omega_2 \leq 367rpm. \tag{5}$$

## 3.2 Momentum Unload Operations

To unload the excess momentum stored in the reaction wheels, commands to perform a series of 50-msec thruster pulses are sent to the spacecraft by the satellite controller. This is procedure is essentially manual, initiated no later than two hours before the first thruster is to be fired ($T_0 - 2Hr$). Currently the satellite controller uses a checklist for performing the procedure which is prepared by an attitude control specialist during the planning phase of the momentum unload.

An abbreviated list of tasks performed by the satellite controller during the momentum unload procedure is as follows:

($T_0 - 2Hr$) – Verify the attitude control specialist has included essential data on the checklist such as the anticipated change in reaction wheel speed, the optimum start time for the dump, and the number and polarity of thruster pulses required. In addition, verify the status of the attitude control subsystem is nominal.

($T_0 - 1.5Hr$) – To achieve maximum thruster efficiency, ensure the catalyst bed heaters on the spacecraft are on. Also, ensure a strip chart recorder used to monitor various parameters during the unload operation is configured properly.

($T_0 - 45Min$) – Check the configuration of the propulsion system and ensure the valve drive electronics are enabled.

($T_0 - 15Min$) – Load the thruster pulse widths into random access memory on-board the spacecraft.

($T_0$) – Transmit a command block consisting of a reaction wheel momentum change command (pre-emphasis) and a command to fire the appropriate thrusters.

($T_0 + 4Min$) – Verify the results of the thruster firing is as expected and determine if more thruster pulses are required.

# 4 System Development

## 4.1 Task Selection

A *task analysis* is a generic name for a set of observational techniques in which the tasks that people have created to perform their jobs are studied by observations and interviews. The tools and information that are used to accomplish the tasks are also noted.

To initiate the TDRSS expert system project, artificial intelligence and human factors specialists from the Intelligent Systems Laboratory conducted a task analysis at WSGT of procedures related to controlling the attitude of TDRSS. The purpose of this task analysis was to identify possible applications of expert system technology for increasing the efficiency of the satellite controllers. The results of this particular task analysis documented:

- Who performs which tasks
- The number and proportion of tasks people perform

- The order in which the tasks are done
- The time it takes to complete the tasks

The approach taken was to conduct interviews with satellite controllers and attitude control engineers in which they were asked to verify and refine 25 flow diagrams of procedures related to the attitude control of the spacecraft. The flow diagrams, initially prepared at the Intelligent Systems Laboratory based on information from the TDRSS Operations Handbook [14], describe who is involved in each procedure, what hardware and software is used, what events make it necessary to execute each procedure, and what the inputs and outputs of the procedures are. The flow diagrams were based on a methodology called Structured Analysis and Design Technique (SADT) [9]. The satellite controllers and engineers were also asked to identify the frequency and time required to perform each procedure. In addition to conducting interviews, approximately twelve hours of observations of the satellite controllers were made in the TDRSS Control Center over three days and two shifts. Figure 1 shows the average number of hours per year a satellite controller spends on the 25 attitude control tasks for TDRSS Flight-3.

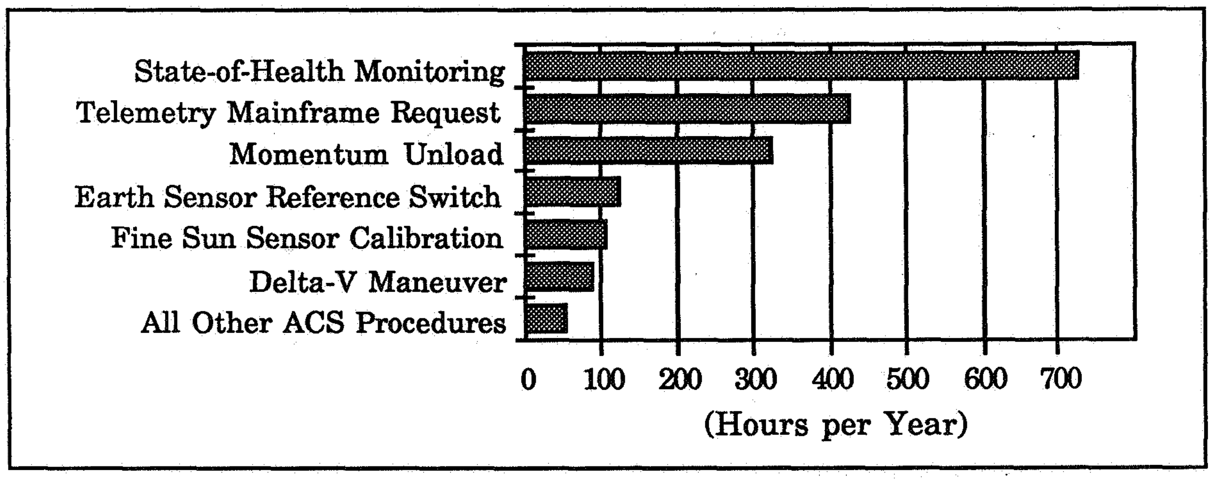


Figure 1: Satellite Controller Time Devoted to Attitude Control of TDRSS

Three categories of tasks were considered for the application of expert system technology: diagnostic tasks, scheduling tasks, and tasks related to the performance of routine procedures. At the request of WSGT management, only the attitude control subsystem of TDRSS was considered for initial expert system enhancement.

In the context of TDRSS attitude control, a diagnostic assistant would diagnose existing and pending attitude related spacecraft component failures. These components include the gyro reference assembly, valve drive electronics, control processor electronics, etc. The task analysis revealed that component failure diagnosis in a system as complex as TDRSS ranges from trivial to extremely complex. The complex diagnostic tasks are currently difficult for teams of experienced spacecraft engineers to solve; therefore, it is not realistic to expect this problem to be solved by an expert system given the current level of technology. Progress on expert systems for complex diagnostic tasks is being made using model-based reasoning, but this work is still in the research stage. Two additional reasons revealed by the task analysis for not pursuing a diagnostic assistant include the discovery that diagnosing spacecraft component failures is primarily an engineering and management rather than a satellite controller function and the discovery that TDRSS components seldom fail.

A scheduling system for TDRSS attitude control would schedule routine procedures in a way to optimize satellite controller time. The task analysis showed satellite controller scheduling to be less of a problem than anticipated. The interaction of WSGT personnel performing satellite controller scheduling and the infrequency of scheduled tasks create an environment in which benefits provided by an expert system would be minimal. In addition, much of the information utilized by scheduling personnel is not currently available on-line and would require extensive modifications to existing procedures before being accessible to an expert system.

An expert system to aid the satellite controller with the planning and execution of specific routine procedures would determine when the selected procedures should be executed, automate parts of the procedures such as the verification of parameter values and the performance of calculations currently done by hand, and guide the satellite controller through the areas of the procedures that require commanding the spacecraft. The expert system would initially encompass a few high payoff procedures. Its knowledge could then be incrementally expanded to include more and more procedures with the goal of eventually covering the entire operation of the spacecraft. The task analysis showed the state-of-health monitoring procedure and the momentum unloading procedure to be good initial candidates for such an expert system. Figure 1 shows state-of-health monitoring to be the most time consuming satellite controller procedure, requiring approximately 730 hours of satellite controller time per year for each spacecraft. The primary function of an expert system aid for state-of-health monitoring would be to intelligently filter alarms. By reducing the number of false alarms with an intelligent alarm filter, the time devoted to this task by the satellite controller could be significantly reduced. Figure 1 also shows that the satellite controller spends about 320 hours per year performing momentum unloads for each spacecraft. In addition, each attitude control engineer spends about 170 hours per year doing the planning required for the momentum unload task. An expert system would be able to completely automate the planning done by the engineer and reduce the time required for the satellite controller to perform the task. The expert system would reduce satellite controller time through the monitoring of telemetry throughout the procedure to ensure the spacecraft is configured correctly and is responding as expected to momentum unloading commands, and by leading the satellite controller through the procedure, suggesting the correct commands to be sent to the spacecraft.

In conclusion, the task analysis suggested an expert system that helps with the execution of routine attitude control procedures would provide the most benefit to the satellite controllers. In addition, such an expert system stands the greatest chance of success given the current state of artificial intelligence technology.

## 4.2 Knowledge Engineering Tools

Knowledge engineering tools for building the TDRSS attitude control expert system consist of a blackboard environment and a production rule extension to the language C. Both of these tools were developed at the Intelligent Systems Laboratory.

### 4.2.1 Blackboard Environment

The blackboard paradigm [12] is appropriate for the solution of problems from domains which, although fairly complex in total, can be partitioned into smaller subproblems and subtasks. It provides an approach to the solution of problems in such a problem domain via the coordination and management of these smaller subproblems and subtasks. The general framework that implements the blackboard paradigm consists of a collection of knowledge sources containing the reasoning and processing facilities required to solve the various subproblems of the problem domain and a blackboard state space that is shared globally by these knowledge sources.

*Knowledge sources* – Each knowledge source contains those reasoning and processing facilities required to solve one of the subproblems of the problem domain. Each is implemented in some manner, such as a rule-based system, function call, process, or asynchronous task, appropriate to the particulars of that subproblem or subtask and to the environment in which they are used.

*Blackboard* – The blackboard state space, referred to either as blackboard shared memory, or simply as the blackboard, provides a data structure for representing the problem-solving state. Each knowledge source has access to the current state of the problem's solution by reading this state space, and may communicate requests for assistance as well as report the results of its solution efforts by writing to this state space.

The blackboard system implementation used to build the TDRSS attitude control expert system consists of two layered subsystems: 1) an object system which is the basis of a shared memory database that provides specialized support for the declaration and the run-time management of control-events associated with shared database elements, and 2) a set of functions and structures that support the association of knowledge sources with these control-events, and the customization and activation of a control loop manager which governs the selection and activation of knowledge sources.

The underlying object system on which the blackboard system is implemented is a general-purpose dynamic object system possessing many of the major features and characteristics of LISP-based object systems such as FLAVORS and LOOPS. The basic objects of the system are its classes and their instantiations. Both of these are characterized by their attributes. Blackboard shared memory is implemented as an object-oriented database that is shared by some collection of knowledge sources as their medium of communicating the global status of a common problem's solution with other knowledge sources.

Within this blackboard system implementation, the coordination and cohesion of knowledge sources is provided by a control loop manager which coordinates the activation of knowledge sources. This manager functions as a mini-operating system and database manager as it regulates the interactions of knowledge sources via the blackboard database.

### 4.2.2 Rule System

Some of the blackboard knowledge sources of the TDRSS expert system are being developed using a simple production rule extension to the language C. Rule systems have proven to be a useful method of modeling the problem solving activity of experts. They are data driven, easily extensible, and understandable by both the knowledge engineer and domain expert. Rule systems are programs consisting of production rules, a global data base, and a control paradigm.

*Production memory* – Production rules are entities made up of an antecedent (also known as an if-part or left-hand-side) which specifies when the rule is applicable, and a consequent (also referred to as a then-part or right-hand-side) which specifies what actions to perform if the rule is executed. The production rules are collectively referred to as production memory.

*Working memory* – The global data base of a production system is called working memory. Working memory provides the context in which antecedents are true (satisfied) or false (unsatisfied).

*Control paradigm* – The control paradigm of a production system is a recognize-act cycle in which rules with satisfied antecedents are identified, a single rule is selected from the satisfied set by a conflict resolution procedure, and its consequent is executed. Identifying rules with satisfied antecedents is referred to as pattern matching.

Production memory in the rule system consists of one or more collections of production rules called rulesets. Rulesets provide a mechanism for grouping rules so the recognize-act cycle can be restricted to a subset of the total collection of rules defined by an application. A production rule antecedent in the rule system consists of a conjunction of clauses. The clauses are partial descriptions of elements of working memory. These partial descriptions are referred to as patterns. An antecedent is satisfied if data exists in working memory precisely matching its patterns. A production rule consequent consists of a C compound statement. This code is executed in the context of the working memory data which satisfied its corresponding antecedent. Specifically, the consequent code has access to variables bound to a user defined subset of the working memory data used in the pattern matching process. Rule consequents consisting of arbitrary C statements is a departure from many other production systems in which both the antecedent and consequent are written in a restricted production system language.

Working memory in the rule system consists of a collection of object instances similar to those mentioned in the blackboard environment description. The blackboard system and the rule system will be integrated such that a region of the blackboard state space may assume the role of working memory.

The rule system recognize-act cycle operates in a forward-chaining mode. In a forward-chaining production system, the process of determining which rules should be in the conflict set is based entirely on the current contents of working memory. The conflict set simply consists of all rules with satisfied antecedents. This is in contrast to a backward-chaining system in which rules are added to the conflict set based on a set of currently active goals. The forward-chaining recognize-act cycle continues until either a solution to the problem is reached, there are no more rules with satisfied antecedents, or a predetermined number of recognize-act cycles have been executed.

A rule system blackboard knowledge source consists minimally of one or more rulesets and C code for initiating forward-chaining on the rulesets. Rule antecedents are compiled by a rule compiler into a discrimination network so that the determination of whether or not they are satisfied may be done with the Rete Match Algorithm, a high performance algorithm for pattern matching [2]. Rule consequents are translated by the rule compiler into C functions to be further compiled by a C compiler.

# 5 System Status and Overview

We are now in a position to explain the architecture of the momentum unload system and how it fits into WSGT operations. We use the blackboard paradigm and encode sections of the monitoring process as knowledge source.

## 5.1 WSGT LAN

The ground station equipment at WSGT is on an Ethernet LAN. Processors on the LAN communicate via *messages* which have a fixed part and a variable part. Each message identifies the time, the source of the message, and the spacecraft. Processors on the LAN operate in a client-server mode and send and receive messages for status values. Messages are identified by a message code.

Our charter for the development of the expert system precluded the possibility of any changes to the ground station software. Thus, our system operates in *wiretap* mode on the Ethernet and receives all of the messages sent and filters out the irrelevant ones. In the initial version, only messages dealing with changed telemetry values and TDRSS commands sent to the spacecraft are examined. Later, as our system expands to handle other facets of TDRSS operations, we may need to examine other messages such as those dealing with user service requests.

## 5.2 Outline of System Components

The major components of the system are show in Figure 2. We now discuss their functions.

*Wiretap* – The Wiretap module examines all incoming packets on the WSGT and reconstructs them into messages. As we have mentioned, messages have a fixed part and a variable part. For example, messages of type 748 contain telemetry values which have changed. The changed values follow the header information and are of variable length.

*Router* – The Router module uses TCP/IP sockets and sets up connections to processes to receive messages. An arbitrarily large number of processes can sign up as clients of the router. Current clients include one telemetry display module per spacecraft and the expert system itself.

*Telemetry Display* – Our telemetry display modules, one for each of the currently on-orbit spacecraft, mirrors the current nominal operating displays now in use at WSGT. After future discussions with the ACS specialists, we expect to add additional display capabilities including trend analysis. The telemetry display was set up outside of the blackboard structure to guard against any performance problems in the display of the telemetry while the blackboard process is running.

*Telemetry Database and the Log/Delogger* – We have defined a flat file and a simple retrieval system to hold telemetry. Adding items to the database is called *logging* and retrieving them is known as *delogging.* We expect that refinements to the system will have to be made over time, so we have allowed the possibility of saving historical telemetry at various time resolutions. For example, a resolution of about 6 hours is needed to plan a momentum unload under current operational procedures, but anomaly resolution is best done if *every change* in the telemetry is available.

## 5.3 User Interface and Status Display

The user interface is based on X-WINDOWS with the Sun Microsystems user interface development tool Guide. Within the structure of the blackboard knowledge sources, there are a a number of display and user interface programs:

*Telemetry Display* – We have mentioned already that display of the telemetry is done outside of the blackboard as shown in Figure 2. The telemetry display shows about 70 telemetry values and has the capability to show another 10 values of the users choosing at any time. We expect that the displays will be modified over time to correspond to specialist preferences.

*Edit Specialist Preferences* – In current TDRSS operations, there is one ACS specialist assigned to each of the spacecraft. The specialist preferences represent information that the expert system can access as part of its expertise and can be viewed as a collection of parameters to control momentum. For example, it might include the actual shift schedule of the specialist so that no momentum unload would be scheduled while the specialist was off-site.

*Display Momentum Unload Plan* – At any given time, there is an anticipated time to do the next momentum unload based on the momentum growth profile of the particular satellite. This display gives a graphical representation of the recent momentum growth and presents in tabular form the time to do the next momentum unload and the sequence of appropriate thruster firings needed to accomplish it.

*Display Momentum Unload Status* – This display identifies each step of the momentum unload and determines which step we are in. We describe this process in more detail in the monitoring section below.

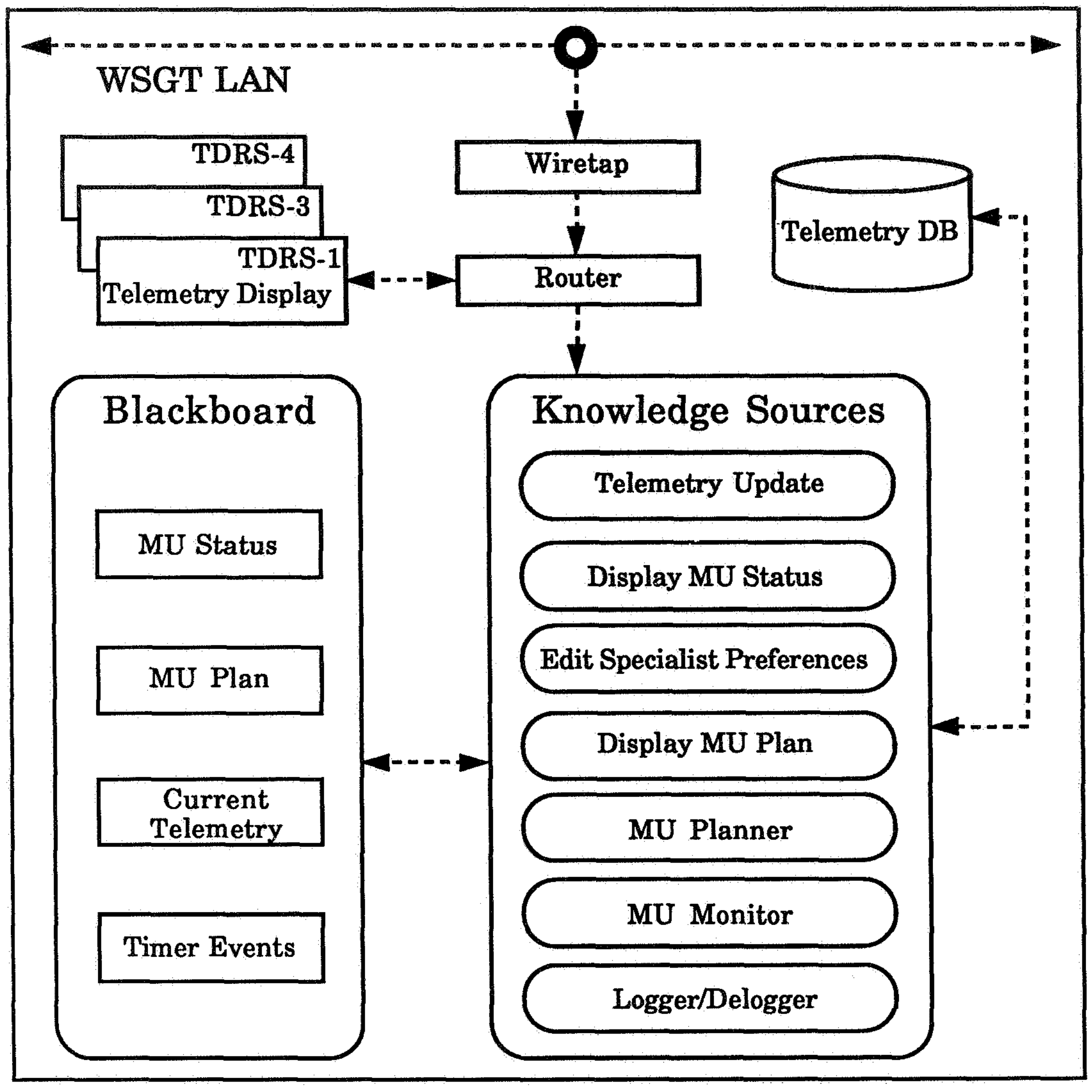


Figure 2: Expert System for TDRSS Momentum Unloading

## 5.4 Expert Momentum Unload Planning

Two currently existing programs are used by WSGT attitude control specialists to plan TDRSS momentum unloads. Both of these programs use the same basic algorithm:

1. Determine the roll and yaw inertial momentum ($H_{x_i}, H_{z_i}$) at the preceding two 0600 and 1800 local spacecraft time (LST) points.

2. Using linear extrapolation, estimate $H_{x_i}$ and $H_{z_i}$ at the next 0600 and 1800 LST points.

3. Compute an average inertial momentum estimate halfway between the 0600 and 1800 LST estimates.

4. Determine inertial momentum target points (typically by accepting them as input from the specialist).

5. Compute the optimal dump time, appropriate thrusters, and the number of thruster pulses needed to achieve the inertial momentum target points.

To help the attitude control specialist visualize the momentum growth of the spacecraft, the existing programs plot roll and yaw momentum in inertial coordinates in three hour increments beginning at the first data point used in the estimation and continuing through the current momentum of the spacecraft. The estimated points are also shown on the plot.

Although these programs are an improvement over the earlier paper and pencil method of planning momentum unloads, the procedure can be fully automated by capturing the knowledge of an attitude control specialist in an expert system and giving the expert system on-line access to spacecraft telemetry. Access to telemetry is achieved by connecting the expert system to the WSGT LAN as has been previously discussed. Attitude control specialist knowledge is being captured in a collection of production rules developed incrementally through a series of interviews between a knowledge engineer and an attitude control specialist.

Access to telemetry gives the expert system the ability to do momentum growth trending. The momentum growth profiles of the individual spacecraft drift slowly over time. By observing the growth profiles over several weeks, the expert system can estimate future inertial momentum data points with a higher degree of accuracy than is currently possible. This capability will enable the expert system to compute target points that will extend the time between unloads; an advantage for spacecraft such as TDRSS in which momentum unloads are not handled automatically by on-board firmware. Human attitude control specialists currently strive to extend the time between unloads using mental models of momentum growth built from their personal observation of the spacecraft over many months.

Another use of momentum growth trending is to enable the expert system to recognize spacecraft maneuvers and erroneous data points. Both of these conditions are suggested by discontinuities in momentum growth. A maneuver (*e.g.* delta-v) is recognized by a shift in inertial momentum followed by a translated momentum growth with the same slope. If a maneuver is detected, the expert system will still be able to accurately estimate future points through the use of the earlier trend. Erroneous data points often occur at the transition between the portions of the orbit in which the spacecraft yaw angle (one of the parameters used to compute inertial momentum) is computed from telemetered values of the fine sun sensors (observer mode), and portions of the orbit in which the yaw angle must be estimated from earth sensor and wheel speed values (estimator mode). These points are characterized by a few discontinuous points followed by a return to the original growth line. Below a specified threshold, the expert system will simply discard erroneous points.

The attitude control specialist also uses scheduling knowledge in the development of a momentum growth plan. For example, the selection of a target point may be influenced by the current day of the week. If a particular target point would cause the critical planning phase of the next momentum unload to occur on a day in which the specialist is not onsite (*e.g.* weekend or holiday), the target point would be adjusted if it were possible to do so and not exceed normal operating ranges. Another example of the use of scheduling knowledge is the use of solar eclipse information to avoid momentum unloads when the spacecraft is running on battery power. The expert system will access a database of this type of information specific to each spacecraft and use production rules to constrain planning decisions.

Although momentum unload planning will be automatic under normal conditions, the attitude control specialist will be alerted by the expert system when momentum growth does not follow the expected trend.

## 5.5 Momentum Unload Monitoring

The momentum unload process has a sequence of steps which are always followed and is defined by a combination of telemetry values and spacecraft commands. Since the monitoring occurs without interaction, *i.e.*, no data can be requested or repeated, the monitor waits for the appearance of telemetry value changes and commands that signal the beginning of a momentum unload and indicate its progress.

**1) Valve Drive Electronics Enabled**

**2) Proper Catalyst Bed Heater On**

**3) Thruster Pulse Width Command Sent**

**4) Proper Thrusters Enabled**

**5) Proper Pre-emphasis Command Sent**

•

•

•

Figure 3: Momentum Unloading Indications

The first few commands for a yaw momentum unload are shown in Figure 3. TDRSS normal mode means that the attitude control is under control of the control program electronics and that the earth sensor is providing feedback to maintain attitude. A momentum unload is best characterized by a thruster firing in TDRSS normal mode. Other maneuvers like a delta-v (change in momentum) are accomplished in earth mode; thus the presence of normal mode telemetry plus the beginning of a thruster firing sequence indicates a momentum unload. In Figure 3, the first steps to initiate a thruster firing include turning on the heaters for the catalyst bed (required for the hydrazine thrusters), turning on the valve drive electronics, and sending commands to initiate a sequence of thruster firings.

We observe these steps under the assumption that all data is provided and is accurate. On the other hand, because of ground station LAN failures and RAM hits (single-event upsets in TDRSS on-board memory), some telemetry values may be missing or corrupted and some commands, although issued, may not reach the spacecraft. The other problem we face is that the messages of type 748 sent along the LAN indicate only changes in telemetry values and if we miss a changed value, there is no way to refresh it in our monitor. However, every hour all telemetry values are refreshed, so our database will become correct over time. We support the development of multiple lines of reasoning to reach conclusions about our position in the momentum unload process. For example, if a command setting the thruster pulse width is noted but the catalyst bed heaters were not turned on, we would assume that a momentum unload is in progress and at the third step even though we do not have every required piece of evidence for this conclusion.

The other feature of the monitoring is integration of the momentum unload plan. The current momentum unload plan is available on the blackboard and can be accessed by the monitor. The monitor can thus estimate the number of thruster firings and the expected time of the momentum unload. As these steps unfold, the plan which the momentum unload plan produces is yet another piece of evidence for observing the plan.

The diagnostic aspect of the monitor includes recommendations and conclusions about the status of the plan. For example, each thruster firing is expected to cause a certain decrement in the reaction wheel speed. Since we know the thruster efficiency and the planned number of firings, the monitor can provide an estimate of the status change in wheel speed.

# 6 Conclusions and Future Work

We have outlined the current status of our system to assist momentum unload operations. It includes a rule-based component which encodes the attitude control specialist knowledge about momentum unloading and a blackboard-based component for monitoring the progress of momentum unload operations. Refinement and integration of the system into WSGT will occur over time.

We expect to evaluate further features of attitude control operations and state-of-health monitoring for inclusion into the system. Furthermore, we will be expanding the coverage of the system to other areas such as power and thermal monitoring.

# References

[1] Carol-Lee Erikson and Peggy Hooker. Tracking & Data Relay Satellite Fault Isolation & Correction Using PACES: Power & Attitude Control Expert System. In *Proceedings of the 1989 Goddard Conference on Space Applications of Artificial Intelligence*, pages 161–168. NASA, 1989.

[2] C. L. Forgy. Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. *Artificial Intelligence*, 19(1):17–37, 1982.

[3] TRW Defence Systems Group. *TDRSS Spacecraft Operations - TMO253*, volume 1. TRW, One Space Park, Redondo Beach, CA, 1985.

[4] K. Howlin, J. Weissert, and K. Krantz. MOORE: A Prototype Expert System for Diagnosing Spacecraft Problems. In *Proceedings of the 1988 Goddard Conference on Space Applications of Artificial Intelligence.* NASA, 1988.

[5] R. Jackson, M. Johnston, G. Miller, K. Lindenmayer, P. Monger, S. Vick, R. Lerner, and J. Richon. The Proposal Entry Processor: Telescience Applications for Hubble Space Telescope Operations. In J. Rash and P. Hughes, editors, *Proceedings of the 1988 Goddard Conference on Space Applications of Artificial Intelligence, NASA Publication 3009*, pages 107–124. NASA, 1988.

[6] Richard M. Keller, Catherine Baudin, Yumi Iwasaki, Pandurang Nayak, and Kazuo Tanaka. Deriving Shallow Rules from Underlying Device Models. In Ethan Scarl, editor, *1989 Workshop on Model Based Reasoning*, pages 124–128. American Association for Artificial Intelligence, 1989.

[7] Denise L. Lawson and Mark L. James. SHARP: A Multi-Mission AI System for Spacecraft Telemetry Monitoring and Diagnosis. In *Proceedings of the 1989 Goddard Conference on Space Applications of Artificial Intelligence*, pages 185–200. NASA, 1989.

[8] D. Lowe, B. Quillin, N. Matteson, B. Wilkinson, and S. Miksell. FIESTA: An Operational Decision Aid for Space Network Fault Isolation. In *Proceedings of the 1987 Goddard Conference on Space Applications of Artificial Intelligence*. NASA, 1987.

[9] D. A. Marca and C. L. McGowan. *SADT Structured Analysis and Design Technique.* McGraw-Hill, 1988.

[10] G. Miller. Artificial Intelligence Applications for Hubble Space Telescope Operations. In A. Heck and F. Murtagh, editors, *Knowledge-Based Systems in Astronomy*, volume 329 of *Lecture Notes in Physics*, chapter 2, pages 5–31. Springer–Verlag, Berlin, 1989.

[11] John F. Muratore, Troy A. Heindel, Terri B. Murphy, Arthur N. Rasmussen, and Robert Z. McFarland. Real-Time Data Acquisition at Mission Control. *Communications of the ACM*, 33(12):19–31, 1990.

[12] H. Penny Nii. The Blackboard Model of Problem Solving and the Evolution of Blackboard of Blackboard Architectures. *The AI Magazine*, 7(3):38–53, Summer 1986.

[13] Frederick W. Rook and Jide B. Odubiyi. An Expert System for Satellite Orbit Control (ESSOC). In Kamal N. Karna, Kamran Parsaye, and Barry G. Silverman, editors, *Expert Systems in Government Symposium*, pages 221–227. The Computer Society of IEEE, 1986.

[14] Contel Federal Systems. *TDRSS Operations Handbook*, volume V. Contel Federal Systems, Las Cruces, NM, Revision A edition, November 1984. WS-0C-0500.

[15] Y. K. Tang and C. R. Wetzel. NASA Ground Terminal Communication Equipment Automated Fault Isolation Expert Systems. In *Proceedings of the 1990 Goddard Conference on Space Applications of Artificial Intelligence*, pages 139–145. NASA, 1990.

N91-22772

# Sharing Intelligence: Decision-Making Interactions Between Users and Software in MAESTRO[1]

Amy L. Geoffroy, John R. Gohring & Daniel L. Britt
Martin Marietta Information Systems Group
M.S. XL4370
P.O. Box 1260
Denver, CO 80201-1260

## ABSTRACT

By combining the best of automated and human decision-making in scheduling many advantages can accrue. The joint performance of the user and system is potentially much better than either alone. Features of the MAESTRO scheduling system serve to illustrate concepts of user/software cooperation. MAESTRO may be operated at a user-determinable and dynamic level of autonomy. Because the system allows so much flexibility in the allocation of decision-making responsibilities, and provides users with a wealth of information and other support for their own decision-making, better overall schedules may result.

## INTRODUCTION

In complex space applications Artificial Intelligence software is often embedded into a large hardware/software/human system to support a target function (Perkins & Truszkowski, 1990), such as fuel loading (Delaune et al, 1985), manifesting (Hankins et al, 1985), ground resource allocation (Durham et al 1990), or scheduling of on-board activities (Ruitberg & Ondrus, 1990). There is a functional division of labor between the various parts of the large system, and AI software can take on any of a variety of roles.

The distribution of *decision-making* responsibilities between AI (or other software) systems and people is of particular interest. A system may be used primarily to automate tasks which are repetitious or boring, e.g. telemetry monitoring (Basile, 1988). Such a system generally provides summarized data, alerts and alarms to a human operator. In this case, a minimum amount of decision-making is performed by the system - it filters and to a limited extent interprets the information for the operator. However, the operator bears the brunt of responsibility for making decisions in this type of system. Sophisticated user interfaces can enhance the overall decision-making performance of the large (hardware/software/human) system by supporting the user with well analyzed, clearly presented and easily accessible information that supports the operator's decision-making process.

An increased role in the decision-making process is realized with advisory AI systems. These systems perform various roles in the decision-making process - some provide occasional critiquing and advice on a user's performance, as with design assistants (Lemke & Fischer, 1990) . Other systems provide constant input and oversight, as in some intelligent tutoring systems (Anderson et al, 1990). Some systems perform most of the requisite reasoning for a task, and recommend courses of action. The user may query the system for lines of reasoning or justifications, and has the final authority to follow or reject the system's advice (Shortliffe, 1984). In these advisory systems, the systems perform much of the reasoning upon which decision-making is based, but the ultimate decision and responsibility rests with the operator.

Few systems fully automate all decision-making. Those that do usually have timing requirements that preclude human interventions, such as process control (e.g. DISPATCHER, described in Kempf et al, 1991), or have strict requirements for autonomous operations, such as some concepts

[1] MAESTRO is a proprietary product of Martin Marietta Corporation.

for the Mars Rover. This is partly due to the brittleness of AI techniques in boundary cases, and partly due to reluctance on the part of potential users to accept totally autonomous decision-making by software systems.

In recent work with our scheduling system, MAESTRO, we have been developing interface support which allows the user to flexibly and interactively vary the division of decision-making responsibility between the user and the system, from use of the system as a decision support tool to operation of the system in a fully autonomous mode. (Fox [1989] also discusses mixed-initiative scheduling, but from a somewhat different perspective). There are two main considerations motivating this approach. The first concerns achieving the best overall system performance - i.e. generating the best schedules possible given both the human and computing resources that can be brought to bear on the problem. The second has to do with user acceptance, i.e. how willingly schedulers will accept the schedules produced by the system.

For many space applications, (e.g. onboard experiment scheduling, ground processing for the shuttle) the quality of the scheduling process has important implications for operational cost, productivity, and crisis avoidance. Scheduling in complex, resource-constrained domains is an extremely hard problem. Currently, realistic scheduling problems are not solved well by either entirely manual or entirely automatic systems. At least in the near term, it does not seem feasible to automate certain aspects of human scheduling performance. This appears to be due to the multiplicity of potentially conflicting goals for a given schedule, and the many and diverse strategies that people can flexibly bring to bear on a particular scheduling problem. On the other hand, efficient scheduling in complex domains is beyond the capabilities of people unaided by computer systems. Even with aiding systems that perform primarily bookkeeping and/or follow standard operations research techniques, scheduling performance can be poor for complex applications.

People and software systems, even artificial intelligence systems, display strikingly different strengths and weaknesses. In many cases the best performance for an AI system is found by *not* performing tasks the same way a person would (e.g. chess programs, Hsu et al, 1990), but by taking advantage of the strengths inherent in computers, such as enormous amounts of memory. People, on the other hand, perform far better than software systems in a variety of ways. Most importantly for scheduling problems, they are far more flexible in their reasoning strategies. By combining the best of automated and human decision-making in scheduling many advantages can accrue. The joint performance of the user and system is potentially much better than either alone. Users can act as high level managers, selecting, enforcing, and switching between global strategies, or tweaking system performance according to their own knowledge and experience. The system, on the other hand, can contribute memory, perform limited brute force searches, do local optimizations, and carry out prespecified scheduling strategies. It is not always the case that joint decision-making will be superior to either fully manual or fully automatic scheduling - e.g. instances where scheduling strategies are easily specified and entirely inflexible can easily be handled by a fully automatic system. However, even in cases where fully automatic and joint decision-making do not produce significantly different schedules, user acceptance for schedules that have been jointly generated will be much higher. There is also potential for migration of user strategies into the software and for increased user directability of the software as it evolves.

Effectively, then, we are trying to create an overall system that gives us the best of both worlds - human intelligence and automated intelligence merged. This paper is a report on what this could mean in practice, and how we are developing a user interface to support this.

The remainder of the paper is divided into four main sections. The first section provides a description of the resource-constrained scheduling problem. The next section describes the MAESTRO scheduling system. In the third section, flexible interactions with MAESTRO supported by the user interface serve to illustrate concepts of user/software cooperation. Examples are given which illustrate the points made above. Finally, we derive some general conclusions from this work.

## THE SCHEDULING PROBLEM

Resource-constrained scheduling is the fixing of activities on a timeline such that those activities may be performed at the times specified by the schedule. This entails the coordination of requisite resources, the availabilities of required ambient conditions, and the interleaving of activities which compete for resources.

Take as an example the astronomical mission last December (1990) onboard the shuttle Columbia, STS-35 (ASTRO-1 and BBRXT). For the astronomical mission itself there were four main instruments, three of which shared a pointing system. The instruments were controlled by both ground and onboard crew, and the ground control was often distributed between NASA centers (MSFC, GSFC and JSC). For any given experiment, at least one instrument was in use, with associated requirements for power, thermal, computer and communications. Additional resource requirements for the same experiment included onboard and ground crew support, sometimes from multiple centers, target (star) visibility, and certain other conditions (e.g. pointing system stability, absence of the South American Anomaly). All of these requirements had to be met *simultaneously* in order to achieve a successful observation. Each observation could be in contention with others for resources such as instruments, crew, pointing orientation, or computational time. So a schedule had to ensure both that 1) the resources for one experiment/observation were fully coordinated, and 2) no resource or other conflicts existed between activities as scheduled. In addition, some observations required coordination between instruments, further complicating the scheduling process. Moreover, the astronomical mission was only part of the overall operation of the shuttle. The experimental program competed with other shuttle activities for limited resources such as crew, computer and communications time.

On a mission such as this, every potential observation minute is precious, so schedule validity and efficiency are paramount requirements. Efficiency is produced by packing the schedule as tightly as possible, which generally means performing as many observations/operations concurrently as possible. Unfortunately, scheduling problems such as this are computationally intractable using pure optimization techniques. For a variety of reasons, artificial intelligence techniques provide the most promising approach to automating the scheduling process (Geoffroy et al, 1990).

Over the last several years we have been exploring the use of AI techniques for resource-constrained scheduling problems. We have developed a sophisticated, intelligent scheduling approach embodied in the MAESTRO system, which has been demonstrated in several prototype applications. The challenge of sharing intelligence between the user and the system has been a primary focus of our recent work.

It is important to emphasize that it is the richness of the representation and reasoning strategies already embedded in MAESTRO that make it possible to support profitable interactions with users. Three main elements are required to support joint operator-system decision-making. First, the system must possess a rich, detailed representation of the domain. This indepth representation is required to supply both the user and the system with the basic information that enables intelligent decision-making. Second, the system must possess sophisticated reasoning strategies. For software to be a useful partner in intelligent decision-making (in contrast to being useful as a decision support tool) it must bring its own useful skills to the table. Third, the system must have mechanisms for sharing its information and reasoning processes with the user. Without these three elements, the system will be a weak partner in the decision-making process. We have deliberately aimed for as much system intelligence and power as possible by emphasizing rich, detailed domain representations, and fairly complex reasoning strategies which exploit the richness of those representations. The approach we have followed - information rich representations with sophisticated reasoning strategies - makes the system suitable for joint intelligent operations with users.

In the next section we describe MAESTRO as an approach to scheduling, and in the following section we turn to how decision-making processes of MAESTRO and the user can be flexibly merged at the user's direction.

## THE MAESTRO SCHEDULING SYSTEM

MAESTRO is a prototype software system, using standard and artificial intelligence techniques. It has been used to explore some of the basics of scheduling, and to examine ways of reducing the computational complexity of the scheduling task while trying to achieve high-quality schedules. It has been under development since 1986. A detailed description of the MAESTRO system is required here to provide a basis for understanding the kinds of interactions a user may have with the system.

The MAESTRO system takes as inputs activity models, a scheduling period, profiles of available resources, conditions profiles, and a list of activities requested for scheduling. The system outputs a timeline of scheduled activities, updated resource availability profiles, simple evaluations of the completed schedule, and a listing of activities' success level by priority.

Activities are modeled as ordered series of subtasks, each of which must be performed in the order specified to accomplish the activity. Temporal relationships which are required between a subtask and any other subtask (within the same activity or in a different activity), event, or absolute time may be specified within the activity model.

**Experiment:** CONTINUOUS FLOW
ELECTROPHORISIS
**Subtasks:**

- Run preparation through sterilization
- Flush
- Power up and get ready
- Run
- Flush buffer solution
- Power down and remove sample
- Extract
- Centrifuge
- Culture preparation
- Incubation
- Separation
- Stain and analyze
- Package and freeze

**Priority:** 1

**Number of runs requested:** 4

**Temporal relations:** Packaging subtask must be completed at least 3 hours before loading of returning materials on the shuttle.

**Soft constraints:** Maximize duration of Run subtask, minimize delay between Extract and Centrifuge subtasks, begin experiment as early as possible.

A)

**Subtask:** FLUSH BUFFER SOLUTION

Parent task: Continuous Flow Electrophorisis
Min duration: 35 min
Max duration: 35 min
Min delay: 0 min
Max delay: 0 min

| | |
|---|---|
| Equipment used: | |
| Fluid handling tools | 1 |
| General tools | 1 |
| EqA09 chamber | 1 |
| Resources required: | |
| Crew | 1 |
| Power | 100 w |
| Heat rejection | 100 w |
| Consumables: | |
| H20 | 10 l |

B)

Figure 1. Panel A shows an example of an activity description, B shows a sample sub-task description

Additionally, each activity has some baseline priority (e.g., life support activities are more important than experimental activities), and may be requested for some repeated number of performances. Any activity that can be modeled as a series of subtasks, each with resource and environmental requirements, can be represented by the system.

Each of the activity's subtasks is in turn represented as an entity with a specified (but perhaps variable) duration, a specified delay (again possibly variable) between it and the preceding subtask, and a specification of the resource and environmental conditions requirements that must be met to perform that subtask. Figure 1A shows a partial description of the information resident in an activity description, and Figure 1B illustrates a partial representation of a subtask for the activity shown in Figure 1A.

The representation of activities allows for the specification of both hard constraints - those which must be absolutely satisfied, and soft constraints - preferences which can be ignored if necessary. So for instance, a particular experiment may be required to occur during daylight hours only (a hard constraint), with a preference for around 10 a.m. (a soft constraint).

To build a schedule, the system:

(1) creates sets of related activities based on any specified temporal relations between activities;

(2) selects an activity set to place on the schedule;

(3) for each of the activities in the set:

(3a) places the activity in a valid slot (i.e., a region on the timeline where all timing, resource, and condition requirements are satisfiable); and

(3b) updates the resource profiles according to the usage predicted for that activity;

and then repeats this process (steps 2 & 3) until no more activities may be validly scheduled.

One of the most important features of the MAESTRO system is the method by which we determine whether and where on a schedule an activity may be placed. On every scheduling cycle the opportunity for each activity is determined. The opportunity calculation specifies all and only those points on the timeline where each subtask in every activity may start and end, considering all resource requirements for each of the subtasks, and the required temporal relations between an activity's subtasks. This means that all possible solutions are represented, and no point is specified that could not be used in a potential solution. We can use this measure for two purposes. The first is to derive from this a secondary measure of how hard it is to schedule an item, given the current schedule and remaining resources. Difficult to place, highly constrained items will have low opportunity, and this information can be used to make intelligent decisions about what to select to schedule on a given scheduling cycle. Second, given that we know all possible start and end times for each subtask, we can use this to make decisions about how to place items on the timeline. It is possible, for instance, to find the earliest or latest possible start for a subtask, or the longest possible runtime for a data collection subtask. This information is critical in much of the intelligent decision-making that the system currently performs, and is the basis on which many more intelligent heuristics may be built.

Note that the ability to identify all possible placements using this method assumes that each activity may be placed independently of other, as yet unscheduled, activities. This becomes an important point in scheduling activities that bear some temporal relation to one another (e.g., cyclically recurring activities, or multiexperiment campaign science projects). Additional scheduling strategies have been implemented to accommodate such relationships so that items constrained in these ways are likely to be successfully scheduled where possible. Britt et al (1990) gives a more complete description of temporal relations management within MAESTRO.

Contingency rescheduling operations are supported in MAESTRO. The system uses information about the current schedule, the projected violation (e.g., a resource deficit), the time of the violation relative to current real time, and characteristics of the scheduled activities to determine what activities need to be altered or descheduled to accommodate the contingency. Heuristics relating to various rescheduling goals help determine which items will be altered. Knowledge about the activities' current states, potential repairs (e.g., interruptability), and performance requirements are used to make deschedule, repair, and/or reschedule decisions. Activities that are in progress during a real-time contingency receive special treatment to synchronize schedule fixes into real-time operations. The system also synchronizes schedule changes with subsystems when the contingencies are real-time or near real-time.

## SUPPORTING THE USER IN JOINT DECISION-MAKING INTERACTIONS

MAESTRO was originally designed to work autonomously, and the description of MAESTRO given above presents MAESTRO as it operates in an autonomous decision-making mode. In this mode, there is little user control beyond some parameter specifications, such as weightings on decision criteria. As the project matured, it became apparent that:

a) MAESTRO occasionally did stupid things, and a little user intervention could go a long way, and

b) experienced schedulers could get a lot of mileage using partial results of MAESTRO's reasoning processes (e.g. identification of opportunity windows), while imposing their own strategies during the scheduling process, and selectively intervening in the scheduling process.

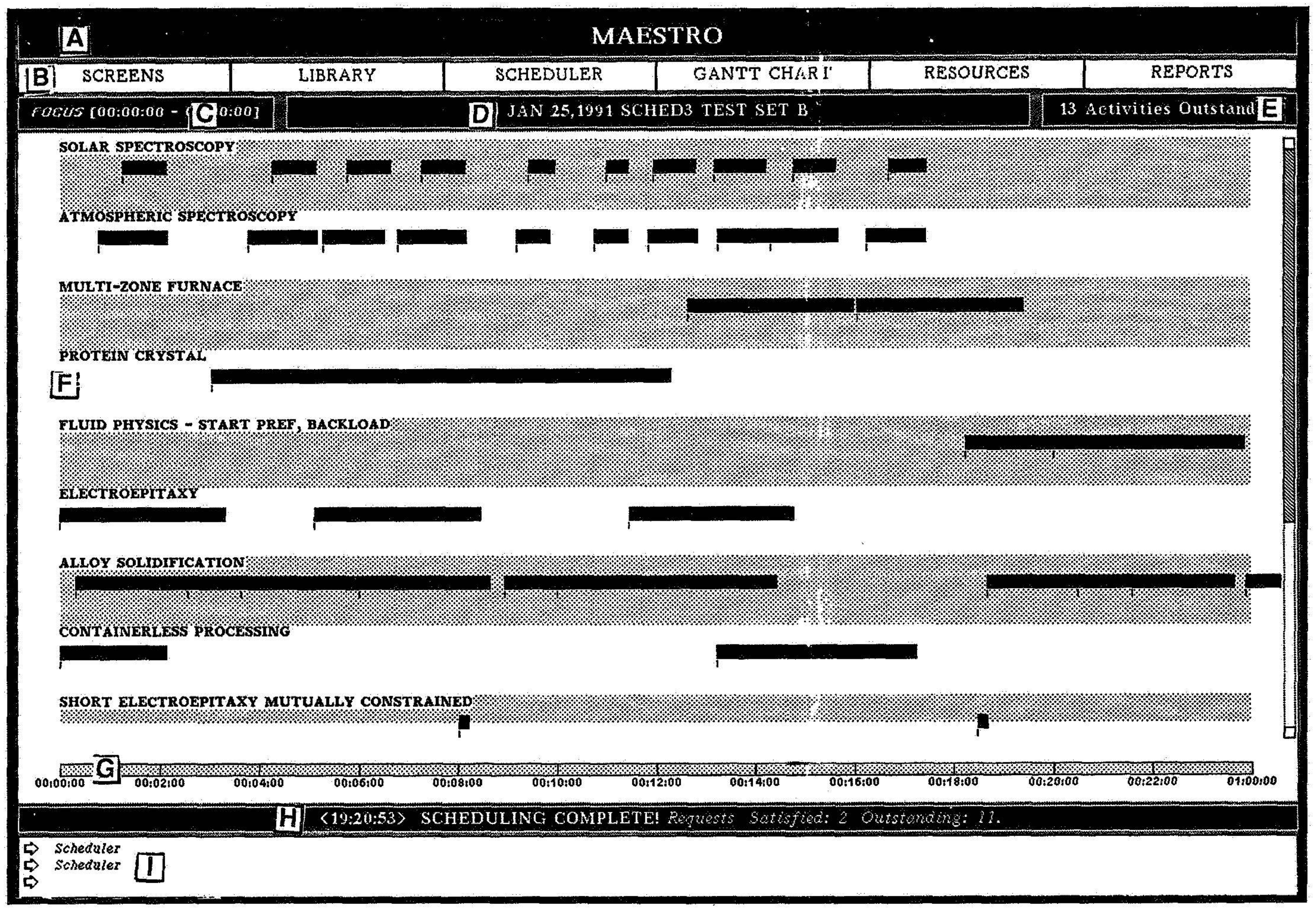
In short, the intelligent behavior of joint user/system decision-making could be a big improvement over their behavior in independent operation. The way to achieve this was to provide the user with the means of selectively and flexibly driving the division of responsibility. The user could then allow the system to perform autonomously when that seemed best, and take over decision-making responsibilities when skilled human judgement would produce superior results.

This required major changes in the user interface and some of the control flow in the system. The user interface must provide support for different allocations of user/system responsibility[2]. This includes support appropriate to each level of interaction, plus mechanisms for transferring decision-making responsibility between the user and the system. Because the system is rich in available information, and in the number of control options available to the user, accessibility of information and control through the user interface is a key issue. Hence, the discussion begins with a description of information accessibility. This is followed by a description of the kinds of support the user receives for different divisions of user and system control. Subsequently, we describe how levels of control can be intermixed and varied during a scheduling session. Finally, an example is given of how this works for an instance of contingency handling.

### *Information Accessibility*

In supporting effective joint decision-making, the access to information is as important as the access to control commands themselves. The structure provided to the user in navigating through the system to access information and control is key to successful overall performance. The MAESTRO user interface provides convenient access to enormous amounts of information and system control. The interface is organized to provide users with guidance and structure without unnecessarily restricting their actions.

[2] Note that there is an underlying assumption that the operator of the system is also a sophisticated scheduler. To participate in profitable joint problem-solving, both the user and the system must bring useful knowledge and skills to the table. This is not to say that the user should be obliged to know much about computers, beyond knowing how to run an applications program.



FOCUS [00:00:00 - 0:00]

JAN 25,1991 SCHED3 TEST SET B

13 Activities Outstand

<19:20:53> SCHEDULING COMPLETE! *Requests Satisfied: 2 Outstanding: 11.*

*Scheduler*
*Scheduler*

Figure 2. MAESTRO screen display from a typical scheduling run. Lettered regions of the schedule screen schow:

| | | |
|---|---|---|
| A) Scheduler Name | D) Schedule Name | G) Timeline |
| B) Menu Bar | E) Scheduler Status | H) History of Scheduler Operations |
| C) Current Scheduler Focus | F) Gantt Chart | I) Command Type-In |

ORIGINAL PAGE IS
OF POOR QUALITY

There are more than 70 commands which a user can access through the user interface. These include both querying functions, such as resource displays, and controlling functions, such as activity selection. The user interface is organized to support easy discovery of, access to and use of the various options. Commands have been topically organized into hierarchical menus. The scheduler screen, for instance, has six major menu headings - "Screens", "Library", "Scheduler", "Gantt Chart", "Resources", and "Reports" (see Figure 2). Each of these provides relevant sub-menus, some of which are themselves topically organized. So, for instance, under the "Library" menu, the items are divided between "Permanent" and "Snapshot" operations. Where a given command requires the user to supply arguments, the system supplies an appropriate dialog prompting the user for the required information.

Each of the menus is context sensitive. The displays only reflect commands which are legitimate to perform given the current state of the system. For instance, you can only retrieve a saved schedule under certain conditions - there must be a schedule already saved, and you can't be in the middle of creating another schedule. The "Library" menu reflects this - it will display the "Retrieve Schedule" option only when the system is not scheduling, and there are items in the schedule library. These menus help to guide the user to appropriate actions given the current state of the schedule and scheduling process.

Many objects on the display are active. Mousing on activity names or bars in the Gantt chart can retrieve a wealth of information about each activity, and parts of the information displays are themselves active. So, for instance, the user can mouse on an activity name to get a list of subtasks, and mouse on a subtask within that list to get a fuller description of that subtask. The potential actions are clearly explained on the mouse documentation line, so that the user can always determine the effect of an action before trying it. Various displays are available which provide information about resources, activities and their sub-components, and general schedule information.

In addition to providing information about a particular schedule, the system also provides for schedule saves, both permanent and snapshot (temporary). Among other things, this allows users to develop a partial schedule, save it, and generate and compare alternative completions of the schedule.

*Different Levels of Decision-making Control*

The user may choose to exert no control over the system beyond selecting a scheduling period and a set of activities to be scheduled during that period[3]. The system will default to a set of standard parameters' values, and scheduling will be performed entirely autonomously using those defaults.

One level of user control involves the user in setting some or all of these parameters. These parameters include:

a) weightings for activity selection in regular scheduling operations (the relative importance of success, opportunity and priority in determining the next item to be scheduled),

b) weightings for activity selection in contingency situations (the relative importance of ten criteria for determining which item to alter or deschedule in a contingency/resource overbooking situation. In addition to success, opportunity and priority, factors important in descheduling, such as whether the activity is already initiated or easily alterable are considered) ,

[3] Note that the activity descriptions (which have been previously specified) contain scheduling directions about both hard and soft constraints. Hard constraints are those which absolutely must be met for the item to be successfully scheduled, and soft constraints represent preferences. MAESTRO always schedules activities fully respecting the hard constraints. In the default mode, MAESTRO uses the model-specified soft constraints to guide placement. Thus, the person who defined the activity is also exerting influence on the scheduler's performance.

c) selection strategies (default is the weighted criteria described in (a), but the user may select a different strategy, such as omitting opportunity calculation for selection),

d) placement strategies (the default is to follow the preferences specified in the activity model, but the user may choose to ignore these preferences), and

e) scheduler focus (the user may instruct the system to only pay attention to a portion of the scheduling period rather than its entire length).

Once these system parameters have been set, MAESTRO can again perform scheduling entirely autonomously. The strategies used in scheduling, while selected by the user, are all part of MAESTRO's repertoire.

Users may take further control of the scheduling process by enforcing their own control strategies in running the system. The mechanism by which the user performs this is by selecting options[4] for:

a) User activity selection - On each cycle the user determines which activity will be scheduled.

b) User activity placement - For a user-specified subset of requested activities the user places each of these activities.

c) User activity deletion - The user may delete specific instances of scheduled activities.

d) Manual altering of resource profiles. This allows the user to increase or decrease the amount of an available resource for any arbitrarily specified portion of the timeline.

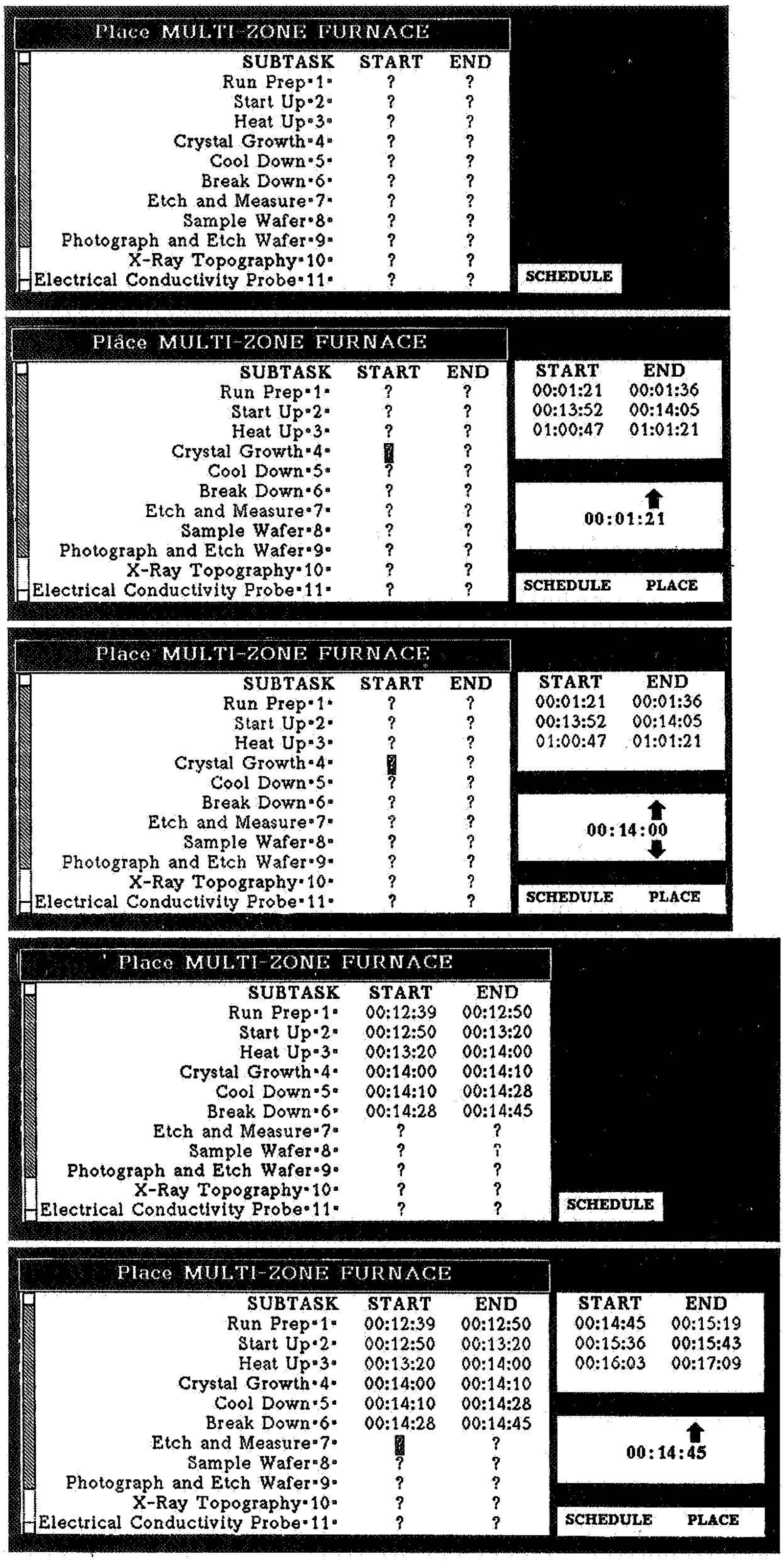
By using these options, the user may take on most of the decision-making responsibility, and MAESTRO then acts more as a decision-aiding mechanism. However, MAESTRO does supply a good deal of support for each of these user operations. MAESTRO provides this support by supplying the user with access to the information the system normally employs in its own decision-making processes. The primary functions used in this way are the system's bookkeeping, constraint propagation, and temporal relations management mechanisms.

a) User activity selection - The system provides a menu from which the user selects an activity to be placed. Only activities which still have opportunity are displayed, so that the user does not try to place an activity which can't fit on the schedule. Additionally, items which are in a related set due to their temporal relations are displayed as a group, so a user will know that all members of that group will be selected for placement on that scheduling cycle. The menu also indicates which activity MAESTRO would select, and the user can accept that option if desired.

b) User activity placement - Recall that activities consist of subtasks with variable durations, and variable delays between subtasks. This allows for flexibility in the scheduling process - more things can get scheduled, and resources can be used more efficiently than if durations and delays are fixed. However, activity placement is much more complex, for the user as well as for the system. To support user placement, MAESTRO provides information about possible placement options based on the previously described opportunity calculation. Whenever an activity designated for user placement is selected for scheduling, an interactor[5] for that activity appears (see Figure 3A). The interactor displays each subtask name and a mouse-selectable icon for each subtask start and end. The user selects the subtask start or end that he wishes to specify, and all possible times for that particular point are provided in the upper right window of the interactor (Figure 3B). The user selects from among

[4] These user options may be used in any combination, e.g. user placement may be used with or without user selection.

[5] An interactor is a window partitioned into subwindows, which provides both information and guides the user through a series of interactions with the system.



Place MULTI-ZONE FURNACE

| SUBTASK | START | END |
|---|---|---|
| Run Prep•1• | ? | ? |
| Start Up•2• | ? | ? |
| Heat Up•3• | ? | ? |
| Crystal Growth•4• | ? | ? |
| Cool Down•5• | ? | ? |
| Break Down•6• | ? | ? |
| Etch and Measure•7• | ? | ? |
| Sample Wafer•8• | ? | ? |
| Photograph and Etch Wafer•9• | ? | ? |
| X-Ray Topography•10• | ? | ? |
| Electrical Conductivity Probe•11• | ? | ? |

A) The interactor presents the user with a list of the activity's subtasks, and icons (the "?"s) to select from to choose the subtask start or end the user wishes to specify first.

Place MULTI-ZONE FURNACE

| SUBTASK | START | END |
|---|---|---|
| Run Prep•1• | ? | ? |
| Start Up•2• | ? | ? |
| Heat Up•3• | ? | ? |
| Crystal Growth•4• | ▮ | ? |
| Cool Down•5• | ? | ? |
| Break Down•6• | ? | ? |
| Etch and Measure•7• | ? | ? |
| Sample Wafer•8• | ? | ? |
| Photograph and Etch Wafer•9• | ? | ? |
| X-Ray Topography•10• | ? | ? |
| Electrical Conductivity Probe•11• | ? | ? |

| START | END |
|---|---|
| 00:01:21 | 00:01:36 |
| 00:13:52 | 00:14:05 |
| 01:00:47 | 01:01:21 |

B) The user decides to begin by specifying the start time for the Crystal Growth subtask. The interactor displays all possibilities (i.e. this subtask can be started anytime between 1:21 and 1:36, 13:52 and 14:05 or 1:00:47 and 1:01:21) in the upper right subwindow.

Place MULTI-ZONE FURNACE

| SUBTASK | START | END |
|---|---|---|
| Run Prep•1• | ? | ? |
| Start Up•2• | ? | ? |
| Heat Up•3• | ? | ? |
| Crystal Growth•4• | ▮ | ? |
| Cool Down•5• | ? | ? |
| Break Down•6• | ? | ? |
| Etch and Measure•7• | ? | ? |
| Sample Wafer•8• | ? | ? |
| Photograph and Etch Wafer•9• | ? | ? |
| X-Ray Topography•10• | ? | ? |
| Electrical Conductivity Probe•11• | ? | ? |

| START | END |
|---|---|
| 00:01:21 | 00:01:36 |
| 00:13:52 | 00:14:05 |
| 01:00:47 | 01:01:21 |

C) The user chooses 14:00 as the start time for the Crystal Growth subtask, using the dial in the middle right subwindow. When the user clicks on "PLACE" (lower right) MAESTRO accepts this as the chosen start time, and propagates the constraints this places on the rest of the subtasks' start and end times.

Place MULTI-ZONE FURNACE

| SUBTASK | START | END |
|---|---|---|
| Run Prep•1• | 00:12:39 | 00:12:50 |
| Start Up•2• | 00:12:50 | 00:13:20 |
| Heat Up•3• | 00:13:20 | 00:14:00 |
| Crystal Growth•4• | 00:14:00 | 00:14:10 |
| Cool Down•5• | 00:14:10 | 00:14:28 |
| Break Down•6• | 00:14:28 | 00:14:45 |
| Etch and Measure•7• | ? | ? |
| Sample Wafer•8• | ? | ? |
| Photograph and Etch Wafer•9• | ? | ? |
| X-Ray Topography•10• | ? | ? |
| Electrical Conductivity Probe•11• | ? | ? |

D) The interactor displays all the start and end times that are fully specified. In this case, by specifying the start time for the Crystal Growth subtask, the user has also fully constrained all start and end times for all the subtasks through the end of Break Down. The user may now choose from the remaining subtasks' starts and ends.

Place MULTI-ZONE FURNACE

| SUBTASK | START | END |
|---|---|---|
| Run Prep•1• | 00:12:39 | 00:12:50 |
| Start Up•2• | 00:12:50 | 00:13:20 |
| Heat Up•3• | 00:13:20 | 00:14:00 |
| Crystal Growth•4• | 00:14:00 | 00:14:10 |
| Cool Down•5• | 00:14:10 | 00:14:28 |
| Break Down•6• | 00:14:28 | 00:14:45 |
| Etch and Measure•7• | ▮ | ? |
| Sample Wafer•8• | ? | ? |
| Photograph and Etch Wafer•9• | ? | ? |
| X-Ray Topography•10• | ? | ? |
| Electrical Conductivity Probe•11• | ? | ? |

| START | END |
|---|---|
| 00:14:45 | 00:15:19 |
| 00:15:36 | 00:15:43 |
| 00:16:03 | 00:17:09 |

E) The user decides to place the start time for Etch and Measure next. The user clicks on the icon, and the system displays the remaining possible times that this subtask may start. The process of user selection and constraint propagation continues until all times are fully specified, or the user directs MAESTRO to complete the process by clicking on "SCHEDULE".

Figure 3. The user places a materials processing experiment, Multi-Zone Furnace, on the schedule. Panels A-E show the sequence of steps the user performs to accomplish this, using the User Placement interactor.

these the desired time for that subtask start or end (Figure 3C), and the system propagates the constraints this imposes on other subtasks' possible start or end times[6] . The interactor is updated, filling in start and end times which have been totally constrained by the user's choice (Figure 3D), and letting the user select the next underconstrained start or end time for specification (Figure 3E). This process continues until the activity placement is fully specified. At this point MAESTRO places the activity and performs the normal bookkeeping associated with activity placement.

User activity placement support is a powerful facilitator for the user. First, it guides the user to valid placement options, so there is not a lot of time wasted in shot-in-the-dark or partially informed guesses about where an activity might fit. The more resource coordination required for an activity, the more critical this facility becomes. Second, the information provided allows users to impose their own placement strategies easily. It is easy to find the earliest possible start time, or to start something as soon as possible after noon, to maximize a data collection subtask duration, or to minimize the delay between calibration and data collection subtasks. MAESTRO is able to provide this support because the information provided to the user is the same as that which the system uses in its own intelligent scheduling. By providing the user with intermediate products of MAESTRO's reasoning processes (in this case opportunity calculation and constraint propagation mechanism outputs) the user is relieved of the burden of operations which are hard or impossible for a person, and left to perform tasks which are more suited to human intelligence.

c) Manual activity deletion - The user may remove any activity from the timeline with a mouse command. A verification menu pops up, to ensure that the user intended to delete the activity (all destructive operations require confirmation as a safety feature). Upon user verification, the system removes the activity from the schedule, updates the resource profiles and does other bookkeeping as appropriate. Further, it checks to see if removal has caused any violations of other scheduled activities' temporal constraints. If so, MAESTRO will remove the violated activities and notify the user of any activity's removal. All activities' opportunities are recalculated to reflect the new state of the schedule.

d) Manual alteration of resource profiles - The system provides a dialog through which the user can alter the amount of a resource available. The system automatically updates the resource as appropriate[7]. The system checks to ensure that no resource overbooking has occurred as a result of this change, and notifies the user if there is a contingency. Finally, it updates opportunity for all activities based on the new resource availability profiles. In this case, the functions performed are primarily bookkeeping, but the end result is that the user is informed of any relevant effects of his actions.

*Intermixing Control*

The division of decision-making responsibility does not have to be fixed at a particular level. Throughout the creation of a given schedule, the locus of responsibility can be flexibly varied. This is achieved by changing parameters, selecting different control options, and selectively accepting defaults during the session. Many of the parameters and control options may be changed while the scheduler is executing. The

[6] For example, before selection of the start time for crystal growth at 00:14:00, the end time for crystal growth could potentially have been at 00:01:31, 01:01: 31, or at any of a large number of now invalid times. MAESTRO takes care of ensuring that only valid placement options remain for user selection.

[7] The actual update method depends on the resource type (e.g. rate-controlled or consumable), projected resupplies, etc. However, since the system already knows which resources are of what type, and appropriate update methods for each type, the system relieves the user of the burden of worrying about these sorts of things.

user may also halt the scheduler at any point in execution[8] and change any parameter or control option, then restart the scheduling process where it left off. Additional flexibility in shifting control is available through the user selection and user placement options. The user selection menu specifies which activity MAESTRO would have chosen next, and the user may accept that default without any deliberation. At any point in the user placement process, the user can allow MAESTRO to complete the placement process, using its normal heuristic strategies. In this case, the user only has to do the work of interest (e.g. specify a particular starting point, or maximizing a subtask duration) and may delegate the rest of the decision-making work to the system.

*Example - Contingency Handling*

An example of a contingency handling interaction serves to illustrate the concept of flexibly directed mixed control. Let us assume the following scenario. A user, Jane, has previously generated a complete 24 hour schedule for a space laboratory mission that is due to be flown tomorrow. There has been some anomaly in the power system that will cause a 25% across-the-board reduction in the power available to the laboratory from 8:00 to 11:00 a.m. during that scheduling period. The scheduling system is notified of this contingency. The system updates resource availabilities, and checks to see if any resource violations have been induced. If there is a problem, an alert flashes notifying Jane that there is a contingency.

At this point, Jane can follow several courses of action. She may: a) choose to ignore the contingency for the time being, and perform other, more pressing operations with the scheduler; b) ask the system to handle the contingency automatically, in which case it will follow the contingency procedures previously described; c) check to see whether the parameters for the contingency selection mechanism are what she desires, optionally reset them, and then have the system handle the contingency automatically; or d) choose to handle the contingency herself.

To handle the contingency herself, she will typically need more information, so she would probably check the times and levels of the power resource overbooking for the schedule period through the "Resource" menu. After examining the overbooking information, she can continue her investigation into the problem, ignore it, or let the system handle the contingency (optionally checking and altering the contingency selection parameters). If she chooses to continue her control, she can check which activities use power during the violation periods, then ignore or automatically handle the contingency, or continue with control. At this point she may have in mind a particular activity or set of activities to delete. She will delete these by simple mouse operations, and MAESTRO will update the schedule to reflect these deletions. It will check to see if there is still a violation of the power constraints, and if so that will be available in a display. Jane will check the display to see if a violation remains, and either ask the system to complete the contingency handling process, continue with her own contingency handling process, or ignore the remaining problems. At various points in her process of deleting and checking the results, she may save her partial results in the snapshot or permanent schedule libraries. This will enable her to perform comparisons of alternative courses of action.

When contingency handling is completed, some of the deleted activities, or some other activities that may not have been able to get on the schedule previously, may now be schedulable. Because the system has been doing all the usual bookkeeping/updating this can be easily checked. Jane can then, if she chooses, initiate a new cycle of mixed control in adding things to the schedule.

There are two main points in this example. First, the user can hand off control to the system at many points during the contingency

[8] The scheduler will actually take the halt instruction and proceed to the next logical stopping place, usually by completing an activity placement in progress, updating the resources and performing other bookkeeping chores, and then halting the scheduling process.

handling process. Second, in her own reasoning process, the user accesses much of the information that MAESTRO typically generates for its reasoning process, and relies on the bookkeeping and constraint propagation mechanisms of the system to complement her own decision-making.

When would Jane take control? When performance would be significantly improved by use of information unavailable to the system or by following reasoning strategies the system does not possess, and only when time is not a critical issue. In real operations there will probably always be information and reasoning strategies that are unavailable to an automated system, and it is up to the user to try to merge these with the support that the automated system can provide. It is up to the system to provide mechanisms that ensure the user can flexibly access as many of its useful processes as possible.

## CONCLUSIONS

Users can access an abundance of raw data and intermediate products of MAESTRO's reasoning processes. They may selectively take over some of the decision-making functions, leaving others to the system. Because the system allows so much flexibility in the allocation of control, and provides users with a wealth of information and other support for their own decision-making, better overall schedules may result.

We have made considerable progress in organizing the user interface to support profitable interactions, but far more could be done. There is still much internal information that either is unavailable or hard to access through the interface. For instance, during contingencies users can find out some of the values for the contingency selection parameters for each activity through the interface (e.g. opportunity, interruptability), but not others (e.g. resource fit). The organization of the parameter values that are available is poor for contingency operations - the information is scattered throughout displays that are typically used for other purposes. Some mixed control that could be supported currently is not. Using contingency handling as an example again, the user cannot currently allow MAESTRO to make only a single deletion and halt, or make a recommendation which the user could accept or reject. We are currently planning enhancements that make more of the system's internal information accessible and to make switching control more flexible, leading to more powerful joint decision-making.

## REFERENCES

Anderson, J.R., Boyle, C.F., Corbett, A.T. & Lewis, M.W. (1990) Cognitive modeling and intelligent tutoring. *Artificial Intelligence* 42(1), 7-49.

Basile, L. (1988) Spacelab data processing facility quality assurance/data accounting expert systems: transition from prototype to operational systems. *Proceedings of the 1988 Goddard Conference on Space Applications of Artificial Intelligence.* NASA Goddard Space Flight Center, Greenbelt, MD, 329-341.

Britt, D.L. Geoffroy, A.L. & Gohring, J.R. (1990) Managing temporal relations. *Proceedings of the 1990 Goddard Conference on Space Applications of Artificial Intelligence.* NASA Goddard Space Flight Center, Greenbelt, MD, 123-135.

Delaune, C.I. Scarl, E.A. & Jamieson, J.R. (1985) A monitor and diagnosis program for the shuttle liquid oxygen loading operation. *Proceedings of the 1st annual Workshop on Robotics and Expert Systems*, Houston TX.

Durham, R., Reilly, N.B. & Springer, J.B. (1990) Resource Allocation Planning Helper (RALPH): Lessons learned. *Proceedings of the 1990 Goddard Conference on Space Applications of Artificial Intelligence.* NASA Goddard Space Flight Center, Greenbelt, MD, 17-28.

Fox, B.R. (1989) Mixed initiative scheduling. *AAAI - Stanford Spring Symposium on AI in Scheduling.* Stanford, CA.

Geoffroy, A.L., Britt, D.L., & Gohring, J.R. (1990) The role of artificial intelligence techniques in scheduling systems. *Telematics and Infomatics.* 17 (3/4), 231-242.

Hankins, G.B., Jordan J.W., Katz, J.L., Mulvehill, A.M., Domoullin, J.M., and Ragusa, J.M. (1985) Empress Expert Mission Planning and REplanning Scheduling System. Mitre Corp. Report M85-33, Bedford, MA.

Hsu, F., Anantharaman, T., Campbell, P. & Nowatzyk, A. (1990) A grandmaster chess machine. *Scientific American*, 263(4), 44-50.

Kempf, K., Russelll, B. Sidhu, S. & Barrett, S. (1991) AI-based schedulers in manufacturing practice: report of a panel discussion. *AI Magazine* 11 (5), 46-55.

Lemke, A.C. & Fischer, G.F. (1990) A cooperative problem solving system for user interface design. *Proceedings Eighth National Conference on Artificial Intelligence (AAAI-90)* Boston, MA, 479-484.

Perkins, D. & Truszkowski, W. (1990) Launching AI in NASA ground systems. *AIAA/NASA Second International Symposium on Space Information Systems.* Pasadena, CA. AIAA-90-5055.

Ruitberg, E. & Ondrus, P. (1990) Lessons learned from the Hubble Space Telescope planning and scheduling system implementation and operations. *AIAA/NASA Second International Symposium on Space Information Systems.* Pasadena, CA. AIAA-90-5039.

Shortliffe, E.H. (1984) Details of the consultation system. In B.G. Buchanan & E.H. Shortliffe (Eds.) *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.* Reading, MA: Addison-Wesley.

# TRANSFORMATION Reborn: A New Generation Expert System for Planning HST Operations

Andrew Gerb
Space Telescope Science Institute[1]
3700 San Martin Dr.
Baltimore, MD 21218

## Abstract

The Transformation expert system (TRANS) converts proposals for astronomical observations with the Hubble Space Telescope (HST) into detailed observing plans. It encodes expert knowledge to solve problems faced in planning and commanding HST observations to enable their processing by the Science Operations Ground System (SOGS). Among these problems are determining an acceptable order of executing observations, grouping of observations to enhance efficiency and schedulability, inserting extra observations when necessary, and providing parameters for commanding HST instruments.

TRANS is currently an operational system and plays a critical rôle in the HST ground system. It was originally designed using forward-chaining provided by the OPS5 expert system language, but has been reimplemented using a procedural knowledge base. This reimplementation was forced by the explosion in the amount of OPS5 code required to specify the increasingly complicated situations requiring expert-level intervention by the TRANS knowledge base. This problem was compounded by the difficulty of avoiding unintended interaction between rules.

To support the TRANS knowledge base, XCL, a small but powerful extension to Common Lisp was implemented. XCL allows a compact syntax for specifying assignments and references to object attributes. XCL also allows the capability to iterate over objects and perform keyed lookup.

The reimplementation of TRANS has greatly diminished the effort needed to maintain and enhance it. As a result of this, its functions have been expanded to include warnings about observations that are difficult or impossible to schedule or command, providing data to aid SPIKE, an intelligent planning system used for HST long-term scheduling, and providing information to the Guide Star Selection System (GSSS) to aid in determination of the long range availability of guide stars.

[1] Operated by the Association of Universities for Research in Astronomy for the National Aeronautics and Space Administration

## 1. Introduction

This paper describes the design and operation of the Transformation expert system (TRANS) which was originally built to interface between the Proposal Entry Processing System (PEPSI) and the Science Operations Ground System (SOGS). When observers submit proposals for use of NASA's Hubble Space Telescope (HST) for astronomical observations, these proposals are entered electronically into PEPSI for syntax checking and rudimentary semantic analysis [Jackson88a]. These proposals need to be processed by (SOGS) which is comprised of the Science Planning and Scheduling System (SPSS), which schedules observations within a week, Science Commanding System (SCS) which generates commanding instructions for an observation, Observation Support System (OSS) which monitors operation of the telescope during the observation and Post Observation Data Processing System (PODPS) which receives and processes science data received from HST. Providing inputs to SOGS requires detailed knowledge that the proposer would not be expected to have. Such knowledge includes the best order in which to perform the observations, how the observations should be grouped to minimize telescope movement and instrument reconfiguration, what extra observations are necessary to support the science requested and attributes necessary to plan, schedule, command and process data for the observations. TRANS was designed to operate on the output of PEPSI, using expert knowledge to provide input to SOGS. The input to SOGS is provided in text form to allow override by an expert.

The original purpose of TRANS has been expanded to allow greater use of its knowledge. It provides input to SPIKE, a knowledge-based long term scheduler for HST observations which determines the optimal assignment of observations to weeks. To aid this process, TRANS also generates requests for the Guide Star Selection System (GSSS) to determine long-range availability of guide stars, needed to insure the pointing accuracy of HST. Trans also generates diagnostic error message to identify observations which will cause trouble for SOGS, SPIKE, GSSS or HST. Figure 1 illustrates the data flow through TRANS.

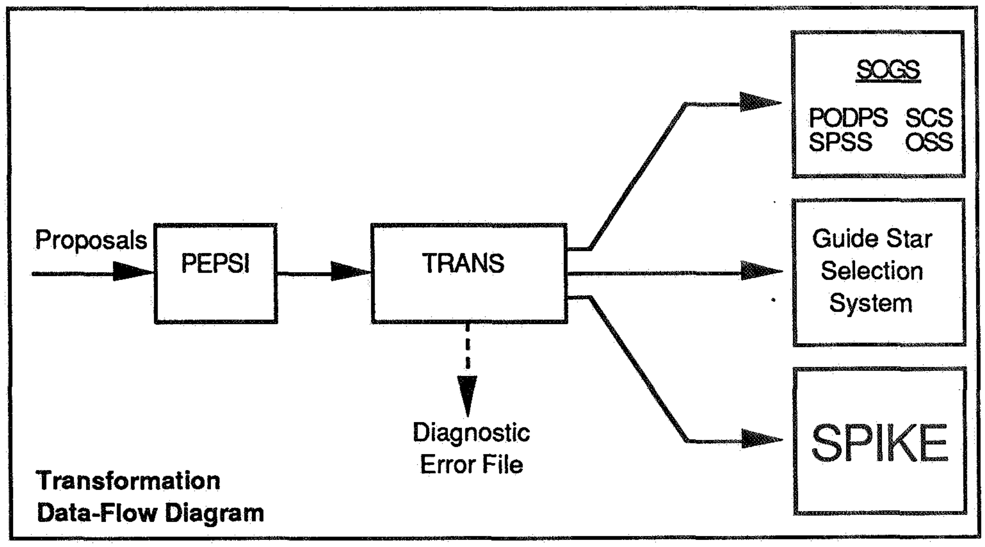


**Figure 1**

## 2. The Knowledge Based Functionality of TRANS

Proposers describe their observing program by dividing them up into *exposures* and entering them onto a form called an *exposure logsheet* [Space89a]. The TRANS knowledge base operates on these exposures taken from a database generated by PEPSI based on the exposure logsheet. Trans performs five basic functions on the exposures it receives as input - ordering, merging, inserting extra exposures needed to support those requested, computing attributes and reporting its conclusions [Gerb90a].

### 2.1 Ordering Exposures

Trans orders exposures based on user-specified constraints. This ordering can be modified (within limits) by SPIKE and SPSS, and is used primarily as a guideline for the grouping of exposures for efficiency purposes. The ordering phase of trans uses technology from the domain of constraint satisfaction problems to insure that all constraints specified by the observer's proposal are satisfied. Details of this process will be made the topic of future publications.

### 2.2 Merging Exposures

The process of dividing exposures into a hierarchical grouping is called *Merging*. During the process of merging, the ordered exposures are grouped into contiguous disjoint sets called *alignments*. Alignments are then grouped into *observation sets (obsets)* and finally, obsets are grouped into *scheduling units*. These groupings define the data structures used by SPSS to schedule HST observations[Miller87a].

Alignments are the tightest grouping of exposures. All exposures in an alignment must use the same HST pointing and orientation, must generate science or engineering data at the same rate and must have only small gaps or interruptions between successive members. Division of exposures into alignments is important for two reasons. First, exposures in the same alignment can be commanded for much more efficient use of spacecraft time. Second, alignments are the basic units of planning for SPSS; SPSS does not schedule on the exposure level.

Obsets are groups of alignments which can be performed without a change in the operating mode of the pointing control system (PCS). HST usually depends on positional monitoring of pairs of stars (called guide stars) to maintain its pointing. A series of alignments can be in the same obset if they all can use the same guide star pair. Alignments which do not need guide stars, such as those that rely on gyros for their pointing control, can also be grouped into the same obset.

Scheduling Units are groups of obsets that SPSS schedules together. When scheduling an obset requires the next obset to be scheduled immediately afterwards, both are placed in the same Scheduling Unit.

TRANS first merges exposures into alignments. It marches down the exposures in the order determined by the ordering phase. For each exposure, it determines if there is an efficiency reason why it should be placed in an alignment with the previous exposure. If such a reason exists, TRANS then looks for a reason why it should not. If this search fails, the exposure becomes part of the alignment containing the previous. If not, a new alignment starts. This process is repeated for merging alignments into obsets, and again for merging obsets into scheduling units.

## 2.3 Inserting Extra Exposures

Once exposures are merged, extra exposures or alignments may need to be inserted. There are currently eight cases where this must be done:

1. When the proposer has requested in the proposal that an exposure be performed multiple times, TRANS needs to copy the exposure.

2. When an alignment needs special commanding or requires monitoring of engineering telemetry by a link to the ground or by tape recorder, an extra exposure is added to the beginning of the alignment to account for the time used in this process and to trigger the appropriate commanding.

3. When an alignment uses the Fine Guidance Sensors (FGS) for science purposes, an exposure must be placed at the end of the alignment to trigger commanding to restore the FGS to its original state and to account for the time used in the process.

4. When an obset uses the Faint Object Spectrograph (FOS), an extra alignment with a single exposure is placed at the end of the obset to trigger commanding to restore the FOS to its original state and to account for the time used in the process.

5. When an alignment sends data to the ground in realtime for analysis, an extra alignment with a single exposure is inserted before the alignment which requires the analysis to allow time for the analysis to occur. It is not necessary to insert this alignment if there is already sufficient time to perform the realtime analysis.

6. When an alignment sends data to the ground in realtime to determine the pointing of a later alignment for target acquisition purposes, a special alignment with a single exposure is inserted before the later alignment to trigger commanding for the position uplink and to account for the time used in this process.

7. When an alignment with more than one exposure is found by TRANS to last longer than a reasonable viewing interval for the target requested, TRANS divides its exposures up into several smaller alignments. It marches through the exposures in the alignment, summing the total time. When the time exceeds the maximum time allowed, a new alignment is created containing the remaining exposures. This process is continued until the remaining exposures comprise an alignment short enough to fit in the time allowed. This process could not occur during the merging phase since merging information is required to compute overhead times.

8. If after the process described in step 7, an alignment containing a single exposure is still too long, that exposure is replaced by several shorter copies of itself, each in its own alignment. First an attempt is made to create two exposures, each roughly half as long as the original, and the overhead time is recomputed. If either is still too long, three are created. This process is continued until either the individual exposures are short enough, or TRANS determines that it is not reasonable to create more exposures.

The eighth example above is an especially complex process, as it is not always possible to break exposures evenly. Many of the instruments on HST require the length of their exposures be an integer multiple of some unit of time (often dependent on the parameters with which the exposures are created). It is up to TRANS to determine the apportionment of time among the new exposures to minimize the number of exposures needed to for all exposures to fit.

## 2.4 Determining Exposure Attributes

Once all exposures, alignments, obsets and scheduling units have been created, there are numerous attributes that need to be computed for each.

### 2.4.1 Overhead Times

TRANS must compute overhead times for all exposures, alignments, obsets and scheduling units. Computing overhead for exposures involves computing the time involved to set up the observations (opening shutter, moving filter in place, etc.), commanding the observations, taking data, reading the data to tape or to the ground, and restoring the instrument to its original state. For some instruments, it is also necessary to determine whether routine calibrations are needed and how long these will take. To do this, TRANS must consider the length of time since the most recent calibration and opportunities for another calibration later in the alignment.

Computing overhead times for an alignment involves summing the times for each of its exposures. If some are required to be parallel with (occurring at the same time as) others, time must be subtracted to take this into account. If the alignment requires a realtime contact with the ground, extra time must be added. Certain instruments require additional time be added to the beginning or the end of the alignment.

Computing overhead times for obsets and scheduling units involve summing up the times for each of their alignments. TRANS determines the number of times the earth has occulted the targets and adds time to compensate. More time must be added to relocate guide stars at the end of each earth occultation. Whenever the target changes, time must be added to complete the Small Angle Maneuver (SAM) required to point at the new target. Also, at the beginning of each obset, time must be added to locate the guide stars used in the obset.

### 2.4.2 TDRS Contact Parameters

If a proposal requires the uplink or downlink of data using the NASA Tracking and Data Relay Satellite (TDRS), TRANS must compute the request parameters. These determine whether the link is an uplink or downlink, whether the link involves science data, engineering data, or both, and the data rate and duration of the contact.

### 2.4.3 Exposure Pointing Information

TRANS records extensive data on the pointing of observations. TRANS decides whether guide stars are necessary, or whether gyros are sufficient to control pointing accuracy. Exposures are marked to be calibration, target acquisition or science exposures. TRANS determines whether the exposure needs to point at a specific target, and if so, whether the pointing can remain the same as the previous exposure. TRANS also computes whether a specific HST roll orientation is required, and exactly where on the HST field of view to position the target.

### 2.4.4 Exposure Scheduling Constraints

TRANS gives SPSS information about scheduling constraints on alignments, obsets and scheduling units. TRANS determines whether they can be interrupted by the earth occulting the target, and what the maximum interruption duration is. TRANS determines, based on constraints from the proposal, whether internal alignments (those which do not point to an external target) can be executed when the telescope is changing its pointing. TRANS determines whether SPSS is free to interleave an obset's alignments those of another. TRANS chooses how large an area to exclude for each alignment when scheduled near the South Atlantic Anomaly (SAA), the region in the southern hemisphere with too high a radiation level to accommodate certain instruments.

### 2.4.5 Commanding Parameters

TRANS computes many of the parameters necessary to command the telescope. The positions of the wheels which place filters, polarizer, gratings and apertures of the instrument field of view are computed by TRANS, determining if wheel motion is required. TRANS tracks of hundreds of commanding values, some specifically requested by the proposer and others it computes. TRANS also keeps track of the states of the instruments at the beginning and the end of each alignment and at what rate data is produced.

### 2.4.6 Computing Timing Relationships Between Objects

TRANS computes timing constraints which exist between alignments, obsets and scheduling units. Such timing relationships arise when an exposure in one higher level object (alignment, obset or scheduling unit) constrains an exposure in another. TRANS computes the minimum and maximum separation of alignments and obsets based on these constraints. When one scheduling unit constrains another, these minimum and maximum separations are noted in the form of a link between the two.

These timing constraints are computed using the same technology as ordering of exposures.

### 2.4.7 Other Values Computed By TRANS

TRANS computes a range of values required by PODPS to determine how to process data once observations are complete. These includes filter, grating, polarizer and aperture usage, as well as camera and spectrograph modes. Trans also performs computations on the target attributes provided by the proposer to make them suitable for use by SOGS.

## 2.5 Reporting the Results of Trans

Figure 2 shows the stages TRANS goes through to build its internal structure.

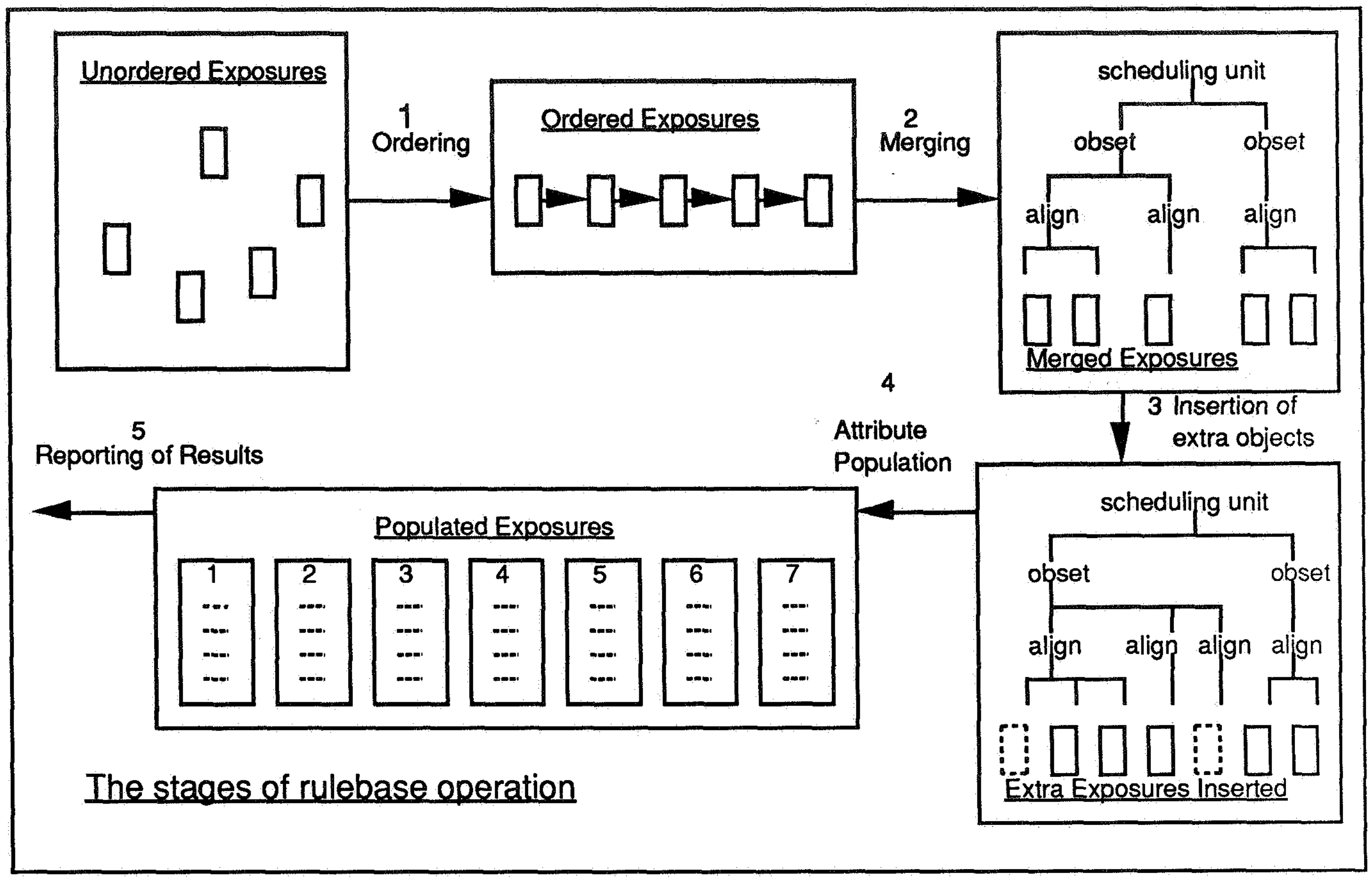


**Figure 2**

After this structure is build, TRANS outputs its conclusions. Though this is chiefly via text file, currently SPIKE is capable of operating in a mode where it can read results directly out of memory.

The largest output product of TRANS is the *assignment file*. Assignment files contain instructions in a database language (IQL or SQL) for populating the Proposal Management Database (PMDB), the chief source of information for SOGS. The assignment file can be edited to override TRANS decisions.

TRANS produces a record of its merging and ordering decisions in the *levels file*. A levels file can be edited and reloaded into TRANS to change the ordering or merging for a second run. In this mode, called *manual merge*, TRANS processes exactly as if it had automatically ordered and merged as specified in the edited levels file.

TRANS generates a *guide star request file* for each obset. Guide Star Requests can be fed into GSSS to determine whether guide stars are available for an obset. The output of GSSS is then used by SPIKE to determine schedulability of observations.

TRANS generates a *proposal auxiliary file* (PAF) for use by SPIKE. When an exposure has been identified by a proposer as conditional, it is recorded in the PAF. This file can be edited when it is determined whether to execute the observation.

TRANS generates a diagnostic file containing a list of problems. TRANS checks for cases it cannot handle and warns about observations that would be difficult or impossible to schedule (such as an alignment which could not be split, but is still too long for an orbit), observations that will present a problem for commanding and exposures that violate constraints of HST.

A series of reports can be generated to allow users to determine what decisions were made. There are several reports that document complicated decisions (such as merging) [Gerb89a].

The outputs of TRANS and their uses are illustrated in figure 3.

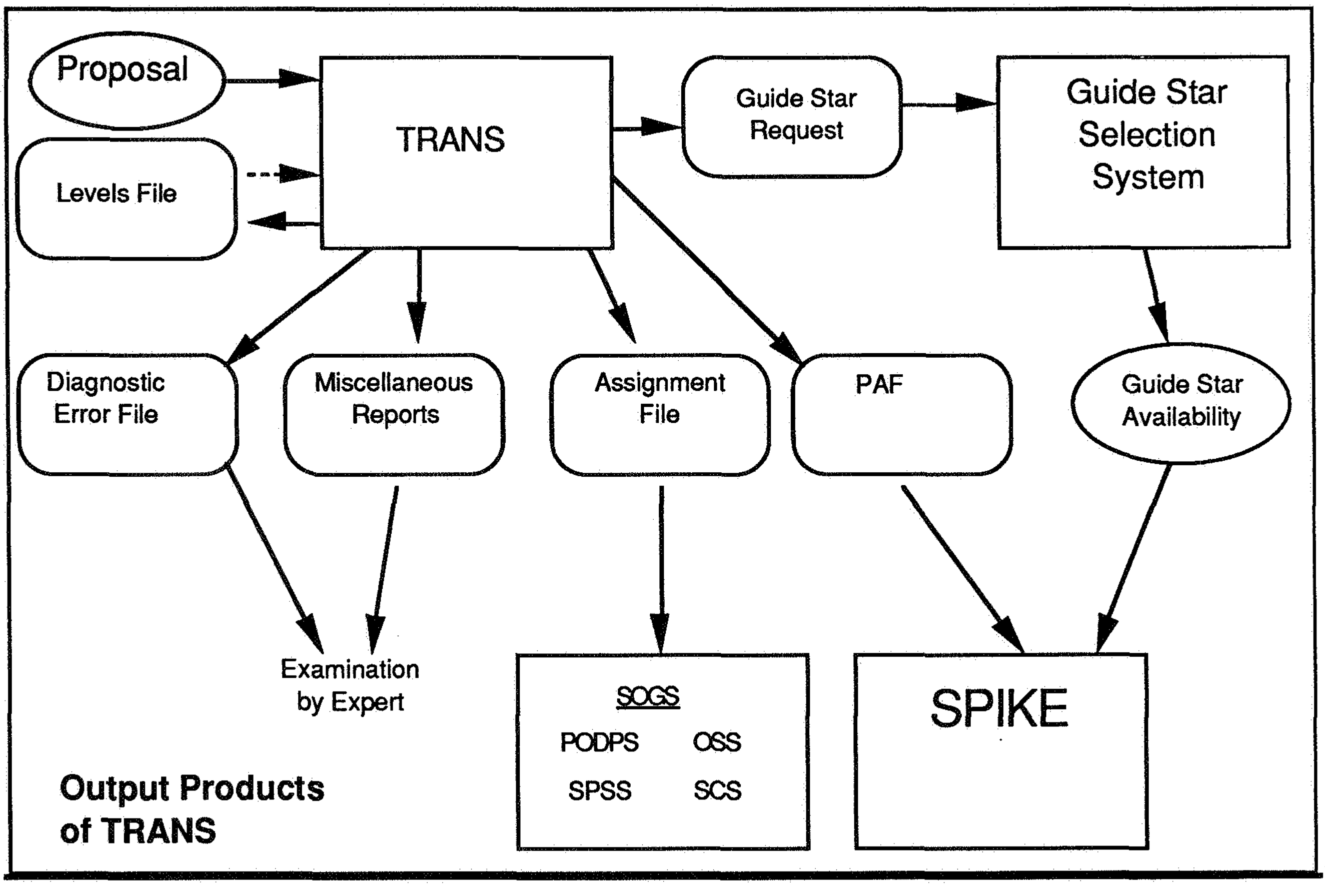


Figure 3

## 3. How TRANS is Implemented

Trans was originally built in 1985 using the OPS5 expert system definition language [Lindenmayer87a, Rosenthal86a, Rosenthal86b]. Although it originally appeared that TRANS solved a simple enough problem to be amenable to the use of OPS5, its subsequent complication and the difficulties inherent in the OPS5 language have forced the conversion of TRANS into Common LISP.

## 3.1. The Implementation of TRANS in OPS5

OPS5 is a declarative language that allows the division of a task into *rules* [DEC89a]. Each rule consists of a left hand side containing conditions under which it should be executed, and a right hand side containing actions to be performed. Permissible actions involve the insertion or deletions of working memory elements (WMEs), the units of data used by OPS5, and the calling of outside functions. OPS5 uses a "bottom-up" order of execution, where a conflict set is accumulated of all rules whose left hand sides correspond with the current state of working memory. A disambiguation algorithm is used to determine which rule in the conflict set to execute. A new conflict set is then computed and the procedure repeats.

### 3.1.1. Problems with Declarative Properties of OPS5

The non-procedural nature of OPS5 made execution difficult to trace. When an undesired result (such as an output product not being correct) was encountered, it was very difficult to tell which rule caused the problem. Procedural languages allow for the placement of debugging output throughout the code to pinpoint the location of a problem. This approach does not work in OPS5, since it is impossible to look at a subset of the source code and determine which rule will execute next. It is necessary to understand the entire rulebase to predict the order of execution of rules. When the OPS5 version of TRANS had grown to over one hundred thousand lines in almost 1000 rules, this became a hopeless undertaking. To debug TRANS, developers were often forced to monitor the execution of rules using the trace facility provided by the environment, stopping execution periodically to determine if the anomaly had occurred. As a side effect, this characteristic of OPS5 forced developers to rely on language specific debugging tools far more than in a conventional language.

### 3.1.2. Deficiencies in the Rule Syntax

The syntax for defining the left hand side of OPS5 rules does not allow full context-free nested boolean syntax. The only condition allowed on the left hand side of an OPS5 rule is a conjunction of assertions of the existence or absence of a WME fitting certain specifications. For example the following condition could be expressed in a single OPS5 rule:

```
Execute if there exists a WME satisfying A and a WME satisfying
B and a WME satisfying C.
```

However, the following condition could not be expressed in a single OPS5 rule:

```
Execute if there exists a WME satisfying A and (a WME
satisfying B or a WME satisfying D) and a WME satisfying C.
```

This condition would have to be implemented by creating two rules with identical right hand sides and the following two left hand sides:

```
1. Execute if there exists a WME satisfying A and a WME
satisfying B and a WME satisfying C.

2. Execute if there exists a WME satisfying A and a WME
satisfying D and a WME satisfying C.
```

If a disjunction is desired in more than one element of the conjunction, there is a multiplicative proliferation in the number of rules needed. For example to express a conjunction of five disjunctions, each of which matched a WME of one of four different specifications, would require 1024 rules!

This scenario was played out over and over during TRANS development. Three major problems resulted from this. First, a requirement that was describable in English in a one line sentence, could translate into a dozen or more rules. Second, this problem often required TRANS developers to devote much time to figuring out the most efficient way to implement simple requirements. Often, no concise way could be found. Occasionally, shortcuts did allow fewer rules, but always at a cost of many hours of planning. Third, the abundance of nearly identical rules provided a constant temptation for developers to use cut-and-paste editing utilities to generate new rules. A minor typographical or logical error could be duplicated scores of times, creating future maintenance headaches.

Because of these problems, it became increasingly difficult to scope work estimates for TRANS projects. Some enhancements took ten times longer than originally expected because a rule explosion was encountered.

### 3.1.3. Lack of a Functional Capability

OPS5 lacks the ability to define procedures and functions. There is not even a macro substitution facility. The normal software engineering activity of providing a library of convenient procedures became impossible when operating in OPS5. One workaround involves the creation of "utility" rules, that populate special WME fields whenever such an operation is desired. This has the unpleasant side effect of greatly increasing the size of WME's and consequently the space required to run TRANS. This workaround does not give the capability within the execution of an OPS5 rule to execute a second rule, returning execution to within the first rule when the execution of the second is complete. OPS5 utterly lacks subroutine definition found in almost all modern languages.

The second workaround involved calling out to a functional language (in our case the C language) to execute function calls. The available interface between OPS5 and C was obscure and poorly documented and differed depending on whether the function was being called within the left hand side or right hand side of a rule and on what kind of data it processed. Calling C subroutines became such a frustrating and error-filled task, so that this workaround was chosen only as a last resort.

The main deficiency in OPS5 utilities was the lack of a string manipulation capability, complicating any requirement which required the examination of text.

## 3.2. The Implementation of TRANS in LISP

Starting in the fall of 1988, TRANS was reimplemented in Common LISP [Steele90a]. Using the LISP macro facility, a small but powerful extension to LISP was written called the transformation command language (XCL). XCL supports a procedural rule syntax and allows abstraction for underlying data structures. A shorthand for using these data structures was developed and a facility for generating simple reports and diagnostic errors was added.

### 3.2.1. Underlying Data Structures

The data structures in XCL allow for different object types. Each object type has a series of properties and keys defined for it [Johnston88a].

Objects were intended to model tuples in a relational database with properties corresponding to the fields of the relation. Definition of properties do not take up any memory space in the data structure unless they are populated with a non-null value. To this end, association lists were chosen to implement the underlying structures, instead of the existing LISP defstruct capability or an object-oriented system such as CLOS.

Keys consist of sequences of properties. For each key, XCL maintains a hash table, matching lists of values (each of which corresponds to a property in the key) to instances of the objects which contain them. In this way it is quickly possible to locate an instance of an object if the values of properties comprising a key are known.

XCL allows a shorthand to specify fields of data structures. It uses the LISP keyword package, choosing the period (.) to separate object instances from their properties. For example `:al.version_num` would correspond to the version_num property of the object denoted by :al. Indirect property reference is also permitted. `:al.exposure.si_used` refers to the si_used property of the object designated by the exposure property of the object :al.

XCL provides forms that allow iteration over a series of exposures. For example:

```
(FOR-EACH :qexposure :ex :do
  (assign :ex.pos_input_fg "N"))
```

sets the pos_input_fg property to "N" for every object of type :qexposure. This capability also allows iterating over all objects that match a certain key or are elements in a certain list.

XCL provides a type checking capability to make sure properties of objects are not referenced that do not exist.

### 3.2.2. Rule Syntax

XCL provides a form for the definition of rules [Johnston89a]. A rule definition is very similar to the defun form in LISP, except that no arguments are necessary. The following rule executes the fragment of XCL code used in the previous section:

```
(define-rule Set-position-input-flag
  "Sets pos_input_fg to N for each qexposure"
  (FOR-EACH :qexposure :ex :do
    (assign :ex.pos_input_fg "N")))
```

This rule is called Set-position-input-flag.

The body of a rule can contain any syntax legal in Common Lisp, as well as the syntactic extension provided by XCL. A facility also exists for defining functions with XCL object type arguments. The following function is an example.

```
(define-function Actual-Alignment-Time ((:al :qalignment))
  "Adds the time_require and acq_time fields of an alignment"
```

```
    (+ :al.time_require :al.acq_time))
```

This function would take an XCL object type of :qalignment as an argument. Run time type checking is performed to insure that the argument is of the proper type.

Rules are executed in the order that they appear in their source file. Source files are grouped into *phases*, which are a series of rules, usually executed together. When a phase is defined, the order of execution of its source files is provided. Phases can also be defined to contain subphases, so that the execution of a phase can involve the execution of one or more lower level phases. Typically, TRANS is executed by invoking a phase containing subphases with all the rules to be executed.

Figure 4 illustrates the relations ship between phases, subphases, source modules and rules.

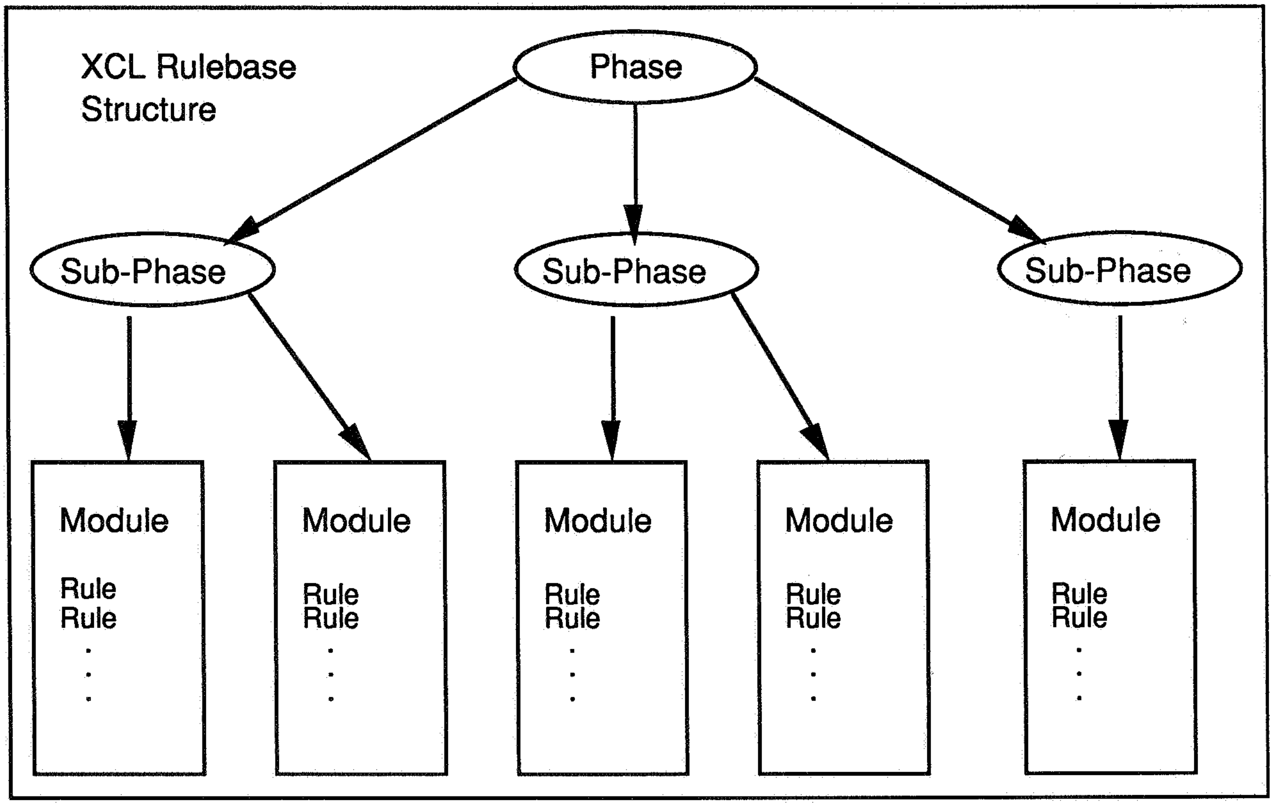


**Figure 4**

### 3.2.3. Other Features of XCL

XCL allows a generator for custom-defined reports. Currently it allows an XCL developer to report over a given object type, outputing one record per instance. Records consist of output fields which corresponds to properties of the object, or computations on it, and may span more than one line. Report fields can be formatted using any of the capabilities available in the Common LISP format statement. Output can be ordered according to keys defined on the output object. Report headings can be specified and one or more title lines included. A planned future enhancement will allow the specification of selection criteria which will exclude any objects from the report.

XCL also provides a table lookup facility where tables are included as part of the XCL source and can be referenced by one or more keys. There is also a flexible diagnostic error generation facility which maintains statistics on the frequency of various problem types.

## 4. Conclusion

Since the conversion of TRANS from OPS5 to XCL, the turnaround time for enhancements has been reduced greatly. As a result, numerous capabilities have been added. The generation of Guide Star Requests, the expanded ability to output diagnostics and the more complex transformations required to create the TRANS output structures are examples of capabilities that the XCL version of trans performs as a matter of course but would have caused major problems in the OPS5 version. Requirements which used to generate numerous rules with over hundreds of lines can now be implemented far more comprehensibly in one rule with only a few lines.

Three major reasons exist for the improved turnaround time. First, the rulebase can be maintained in a modular hierarchical way, making it much easier to understand the order of execution of rules and phases. Second, its procedural nature facilitates debugging of the rulebase - an anomaly can be systematically pinpointed without lengthy tracing of execution. Third, the nature of XCL and its LISP underpinnings have allowed TRANS developers to create a rich functional base, providing all the software engineering benefits of code reusability.

We are now finding that non-developers are occasionally able to peruse the TRANS rulebase and determine its functionality, reducing the need for developers to perform a user support function. This activity was strongly encouraged while the OPS5 version was being maintained, but did not occur in practice until TRANS was reimplemented using XCL.

Three major lessons lessons were learned from this project. First, that as an expert system definition language, OPS5 has deficiencies which intensify as the problems it is applied to become increasingly complex. Second, that it is worth considering a procedural knowledge representation - that such a representation is easy to build, comprehend and modify. Third, that Common LISP provides an ideal platform for such a procedural expert system shell because of its versatility, extensibility and wealth of predefined utilities.

The successful re-implementation of TRANS has made it one of the most flexible elements of the HST ground software systems. Because it provides the "bridge" between observing programs and flight operations, it provides the key point of leverage when changes in observing strategies arise. Work is underway to define how TRANS can further optimize the efficiency of HST, which should make available hundreds of hours per year of valuable observing time that would otherwise be lost, Finally, because of the extensive observation checking performed by TRANS, plans are being mde to provide a distributable version of TRANS that could be used by astronomers at their home institutions to help prepare observing programs

## Acknowledgements

Implementation and maintenanceof the OPS5 version of TRANS involved Don Rosenthal, Kelly Lindenmayer, Jim Sims, Bob Jackson and Andy Gerb. Implementation of the LISP version of TRANS involved Mark Johnston, Bob Jackson, Kelly Lindenmayer, Jim Sims, Andy Gerb, Mike Rose and Ron Henry. Requirements definition for TRANS was aided by Sid Parsons, Doug McElroy, Eliot Malumuth, Mike McCollough, Jim Roberts, Jim Mainard and John Baum.

## References

DEC, 1989, "VAX OPS5 User's Guide", Digital Equipment Corporation [DEC89a].

Gerb, A., 1989, "The Transformation User's Manual", SPIKE Technical Report Number 1989-13, Advance Planning Systems Branch, Space Telescope Science Institute, [Gerb89a].

Gerb, A., 1990, "Transformation Design Manual", SPIKE Technical Report Number 1990-2, Advance Planning Systems Branch, Space Telescope Science Institute, [Gerb90a].

Jackson, R., Johnston, M., Miller, G., Lindenmayer, K., Monger, P., Vick, S., Lerner, R. and Richon, J. 1988, "The Proposal Entry Processor: Telescience Applications for Hubble Space Telescope Science Operations", Proceedings of the 1988 Goddard Conference on Space Applications of Artificial Intelligence [Jackson88a].

Johnston, M., Lindenmayer, K., Sims, J. and Gerb, A., 1988, "Transformation System Programmer Manual", SPIKE Technical Report Number 1988-7, Advance Planning Systems Branch, Space Telescope Science Institute, [Johnston88a].

Johnston, M., Lindenmayer, K., Sims, J. and Gerb, A., 1989, "Transformation Script Programmer Manual", SPIKE Technical Report Number 1989-8, Advance Planning Systems Branch, Space Telescope Science Institute, [Johnston89a].

Lindenmayer, K., Vick, S. and Rosenthal, D. 1987, "Maintaining and Expert System for Hubble Space Telescope Ground Support" in Proceedings of the 1987 Goddard Conference on Space Applications of Artificial Intelligence and Robotics [Lindemayer87a].

Miller, G., Rosenthal, D., Cohen, W. and Johnston, M. 1987, "Expert Systems Tools for Hubble Space Telescope Observation Scheduling" in Proceedings of the 1987 Goddard Conference on Space Applications of Artificial Intelligence and Robotics, reprinted in Telematics and Informatics, 4, 301-311 [Miller87a].

Rosenthal, D., Monger, P., Miller, G., and Johnston, M., 1986, "An Expert System for Ground Support of the Hubble Space Telescope", Proceedings of the 1986 Goddard Conference on Space Applications of Artificial Intelligence and Robotics [Rosenthal86a].

Rosenthal, D., "Transformation of Scientific Objectives into Spacecraft Activities", 1986, Expert Systems in Government Symposium [Rosenthal86b].

Space Telescope Science Institute, 1989, "Hubble Space Telescope, Phase II Porposal Instructions", Space Telescope Science Instute [Space89a].

Steele, G., 1990, "Common LISP, The Language, Second Edition", Digital Press [Steele91a].

N91-22774

# USING C TO BUILD A SATELLITE SCHEDULING EXPERT SYSTEM: EXAMPLES FROM THE EXPLORER PLATFORM PLANNING SYSTEM

David R. McLean and Alan Tuchman
Bendix Field Engineering Corporation

William J. Potter
NASA/Goddard Space Flight Center

## ABSTRACT

Recently, many expert systems have been developed in a LISP environment and then ported to the real world C environment before the final system is delivered. This situation may require that the entire system be completely re-written in C and may actually result in a system which is put together as quickly as possible with little regard for maintainability and further evolution. With the introduction of high performance UNIX and X-windows-based workstations, a great deal of the advantages of developing a first system in the LISP environment have become questionable. This paper describes a C-based AI development effort which is based on a software tools approach with emphasis on reusability and maintainability of code.

The discussion starts with simple examples of how list processing can easily be implemented in C and then proceeds to the implementations of frames and objects which use dynamic memory allocation. The implementation of procedures which use depth first search, constraint propagation, context switching and a blackboard-like simulation environment are described. Techniques for managing the complexity of C-based AI software are noted, especially the object-oriented techniques of data encapsulation and incremental development. Finally, all these concepts are put together by describing the components of planning software called the Planning And Resource Reasoning (PARR) shell. This shell has been successfully utilized for scheduling services of the Tracking and Data Relay Satellite System for the Earth Radiation Budget Satellite since May of 1987 and will be used for operations scheduling of the Explorer Platform in November of 1991.

## INTRODUCTION

The issue of "doing Artificial Intelligence (AI) in C" has been a topic of debate for a number of years now (Schildt, 1987). The primary motivation of this paper is not to demonstrate that it is possible to do AI in C but to demonstrate that there are definite advantages of doing AI in C. Because traditional approaches of software development (waterfall) have not emphasized its reusability, the products of this approach can only be utilized within a narrow range of applications. This is because the waterfall model does not accommadate the sort of evolutionary development made possible by rapid prototyping capabilities and forth-generation languages (Booch, 1991). Recently, NASA has been taking the software reusability issue seriously (Truszkowski, 1989) and there are those who argue that software reuse should be at the heart of the strategy for software maintenance (Longstreet, 1990). A related issue of concern is the need for software to accommodate change (Watson, 1990).

Because reusable software must accommodate changes in the desired behavior through easy reconfiguration, it also ensures that it is to some extent maintainable through the reconfiguration process. However, reusable software must also be fully integrated so that components can be added or deleted easily. Characteristics which improve maintainability include: use of a standard high-level language, modularity and standard coding conventions, which use meaningful names (Longstreet, 1990). Recently, object oriented languages have gone a long way toward allowing the software engineer to obtain the reusability goal.

Object oriented software development has evolved from the user interface technology which is often associated with AI (Goldberg, 1984). The availability of today's high performance workstations has allowed the software engineer to take advantage of some of the AI technology and put it to practical use. To the software engineer, AI technology is just another set of tools available to implement the requirements which eventually accomplish the desired software goals. However, it is easy to imagine the process of converting a LISP-based system to a C-based system as one being done as quickly as possible, without regard to the evolution of the C-based system. In any case, it is a major undertaking (Martin, 1990). Relying on vender support for modifications to an off-the-shelf AI shell may be undesirable and maintaining software which is composed of a mix of different kinds of languages is expensive (Schildt, 1987). Other developers and users of AI technology at JPL (Durham, 1990) have reported similar experience with software tools. To be more useful to the software engineer, AI tools should be part of an integrated set of software tools. Therefore, if a team of software engineers is to take full advantage of the new "AI technologies" it is desirable that the AI tools be directly accessible and written in the same language as the current tools.

Because maintaining software usually includes responding to new requirements and hence support of software evolution (Booch, 1991), the software reusability issue is even more important. Ideally, the software maintenance engineer will use existing software tools to modify a given system and change its capabilities. Therefore, the software tools should become part of the language through which new requirements are implemented. Because all software must eventually change or become obsolete, this kind of extensibility should be a primary goal of all software development projects. Software engineering teams which utilize this approach must become intimately familiar with existing software tools and libraries. This takes time and experience because gaining a working knowledge of the existing tools is similar to learning a new language. However, once this is accomplished, the engineers are in a position to develop systems in a fraction of the time that would otherwise be required. Thus, managers need to allocate time for new members of a software reuse team to learn the "new language" and also to value this expertise once it becomes available. There is a world of difference between an off-the-street C programmer and one who has learned to utilize software tools. It is important that a major effort toward this end be made so that generic software tools and reuse methodologies can be identified and utilized.

Getting started with the software tools approach may require that developers re-think some of their development paradigms because initial development may proceed from the bottom up. Some bottom up development is required because the lower level tools must exist before they can be utilized. Thus, the developers need to learn to think in terms of using and designing for reuse. This also means that software managers need to allow for reuse development and note that there is no need to generate an entire system from scratch. Also, because reusability developers will be using tools written by others, they will require some of the traits of the traditional maintenance engineers; humility and adaptability to the style and ideas of others (Parikh, 1986). With time, many of the distinctions between developers and maintainers may disappear.

This paper describes some of the development effort which has resulted in generic software tools, which include AI technologies, for use in solving scheduling problems. These tools are written in the C programming language (Kernighan, 1978) with an emphasis on object-like development methodology. C was chosen because of the primary maintenance goal of portability. When C++ (Stroustrup, 1986) class libraries become generally available (and reasonably standard), these tools will be re-written to takeadvantage of full fledged object oriented development methodology. The emphasis here will be on integrated AI tool development with examples which demonstrate how AI technology can be utilized with a traditionally non-AI language, such as C. Readers who are not interested in the implementation details may skip those parts without loss of continuity of the general methodology description. On the other hand, the detail reader will note that many of the AI paradigms which seem so exotic to the uninitiated can be implemented in a straight-forward manner.

## GETTING STARTED

In 1985 a group of software engineers from Bendix Field Engineering Corporation, called the Interactive Experimenter Planning System (IEPS) group, were tasked with investigating AI tools and techniques to be utilized for a satellite planning system(McLean, 1987). The task started by looking for tools which might be useful, such as the language support libraries and other software currently available. Eventually, the IEPS software engineers created libraries for file I/O, string manipulation, date and time conversion and user interface tools. These user-defined libraries were written on top of the more or less standard language support libraries and have evolved continuously since their initial creation. The IEPS application developers, in turn, utilized these user-defined libraries (tools) to create prototypes and applications such as the Earth Radiation Budget Satellite System (ERBS) Tracking and Data Relay Satellite (TDRS) contact planning system (McLean, 1987).

For those readers who are farmiliar with LISP but not with C, the following example will demonstrate how some of the behavior of LISP can be simulated in C. In particular, this example shows how to simulate some of the behavior of the LISP primitives CAR and CDR. First consider a string which contains three tokens as follows:

**first second third**

In LISP, the first token is obtained by invoking the string with CAR and the remainder are returned by invoking CDR.

```
(CAR (first second third)) —> first
(CDR (first second third)) —> (second third)
```

Now consider a module written in C called "get_tok" which takes a pointer to a character string as the first argument, a string token buffer as the second argument and a character delimiter as the third argument. Get_tok also returns a pointer to the remainder of the string:

```
sptr = "first second third";
sptr = get_tok(sptr,token,BLANK);
```

Before invocation:

```
sptr —> "first second third"
```

After invocation:

```
token —> "first"
sptr  —> "second third"
```

Because get_tok returns a NULL token when the end of the string is reached, it also provides iterative control as follows:

```
for(sptr =get_tok(sptr,tok,BLANK);
    *tok; /* while tok not empty */
    sptr = get_tok(sptr,tok,BLANK))
    do_something(tok);
```

All of the tools described in the subsequent sections utilize **get_tok** for string processing.

## USER INTERFACE TOOLS

The IEPS user interface tools were designed to be used independent of the application and to be as portable as possible between the PC and workstation hardware platforms. The bulk of these tools reside in a library called MEXLIB (Menu-based EXecutive LIBrary) (NASA-GSFC, 1988) which was designed to utilize an AI technology called Menu-based Natural Language Understanding (Tenant, 1983). These tools allow the application developer to describe the grammar, through which the user interacts with the application, in terms of menus, forms and other user interface objects (widgets). Thus, an application need only read the grammar file in order to know how to interact with a user and invoke the appropriate objects for command line building. Other user interface tools, such as the Transportable Applications Environment (NASA-GSFC, 1990) and others which utilize X Windows MOTIF or OpenWindows, create C source code which then must be compiled and linked to the application. MEX technology avoids this by dynamically creating the objects which are specified by the grammar file.

The LISP DEFSTRUCT data abstraction mechanism allows the user to create data structures. C also provides this capability and MEXLIB is built on these structures. When a user interface grammar file is read, MEX tools dynamically allocate these internal structures and fill in the slots of information specified by the file. For example, if the internal structure is of type menu then the options of the menu are read into a linked list and the appropriate interactive widget is assigned to the method slot. As each MEX structure is built, it is put into a hash table so that it can be looked up quickly by name. Once all the MEX structures have been built, the structure whose name is "main" displays itself to the user and initializes the interaction. After the "main" object has completed its interaction with the user it adds information to a command line which is then parsed. The parser examines each token in the command line and uses a depth first search to "expand" those tokens which match MEX structure names. Expansion is done by invoking the appropriate object which adds more information to the command line.

In a way similar to deriving new classes from base classes, C lets the developer derive new structures from more primitive structures. A simplified MEX data structure can be built upon two other structures, **list** and **form**. List represents a linked list of character strings and is used to hold the options of a menu or the default values of a form. **Form** represents a template with fields to be displayed to the user. **Form**, in turn, is built upon another structure called **form_element**. Each form element has a field name, value, row and column information. **Form** uses an array of form elements to represent the various fields on the form, a template (character page) which is displayed to the user and an index which represents the current field being processed. Examples of the **list**, **form_element** and **form** data structures are given below:

```
struct list {
char *line;
struct list *next;
};

struct form_element {
char *field_name;
char *value;
int row, column;
};

struct form {
struct form_element f[MAXF];
char *temp[MAXLINE];
int current_field;
};
```

In addition to the these structures, the **mex** structure also contains the name of the structure, a title, the menu option selected, the row and column position and a pointer to the interface widget to be invoked.

```
struct mex {
char *name;
char *title;
int i,j;
char *selected;

struct list *list;
struct form *form;

char *(*interface)();
};
```

As a mex grammar file is read, mex data structures are dynamically allocated by invoking mexalloc which uses the standard C library malloc tool. Some of the members are then set to default values until more detailed information is read.

```
struct mex *mexalloc()
{
struct mex *object;
  object = (struct mex *) malloc(sizeof (struct mex));
  object->name = NULL;
  object->title = NULL;
  object->list = list_alloc();
  object->form = NULL;
  object->i = object->j = EMPTY;
  object->selected = NULL;
  return(object);
}
```

Notice that the **list** member invokes a user defined **list_alloc** to allocate its initial dynamic space but that the **form** member is set to NULL until it is known that it will be used. (Forms use the **list** structure for default values but simple menus do not use the form structure.) Because dynamic memory is allocated only on an as needed basis, it is conserved.

Much of the work of parsing the MEX grammar file is accomplished by use of the **get_tok** tool. However, once the mex structures have been built and put into the hash table, the main MEX parser can be invoked to build command lines. A simplified version of the MEX parser is given below and described in the following paragraph:

```
char *parse(line)
char *line;
{
char head[80];
char *tail;
char *select;
    if(!*line)
        return(NULL);
    tail = get_tok(line,head,BLANK);
    if(object = mex_get(head)) {
        if(object->l->next->line)
            select = object->interface(object);
        else
            select = object->l->line;
        object->selected = select;
        parse(select);
    }
    else
        add_tok(head);
    return(parse(tail));
}
```

**Parse** is given a character string (**line**) and if it is empty, the value NULL is returned. Otherwise, **get_tok** is invoked to obtain the first token in the string (**head**). Next, the value of **head** is looked up in the hash table to see if it is the name of a mex structure. If so, then the structure (**object**) is retrieved and its linked list is examined to see of there is more than one option (which would require user interaction). If this is the case, then **object's** user interface is invoked to return the option selected. (In the case of a form, all fields are returned.) Once the option has been selected, a pointer to its value is placed in the "selected" slot of **object** and **parse** is invoked again (depth first) with that selection. If **head** is not the name of a mex structure then it is added to the command line being built by invoking **add_tok**. Finally, **parse** is again invoked on the remainder of the original string (**tail**).

## AN INFERENCE ENGINE

The inference engine developed by the IEPS group is called the Transportable Inference Engine version 1 (TIE1), (McLean, 1986). TIE1 utilizes MEXLIB tools for its user interface and is frame based (Minsky, 1975). Each frame represents a goal or concept which has a default value and a value which is to be sought by application of the rules of inference associated with the frame. The attributes which are referred to in the rules must be specified in the frame attribute list. Thus, frames consist of a frame name, a value, a default value, an attribute list and a rule list. A TIE1 Knowledge Base (KB) consists of a set of frames, one of which represents the goal and the remainder which represent subgoals.

Each simple rule represents a hypothetical instance of the goal or concept and is composed of a rule name which represents a potential value for the frame and attribute-relation-value triplets. For example:

**N_eyes lt 8**

(The number of eyes is less than eight.)

The attributes which make up the rules may be primitive (not decomposable) or they may represent other frames. Primitive attributes obtain their values by interacting with the user or by querying data bases. When a KB is to be used interactively, the KB engineer can specify the MEX-style user interfaces to be utilized for each attribute. Decomposable frame attributes obtain their values from the inference rules associated with its frame and thus represent the backward chaining component of the TIE1 architecture. Complex rules have additional attributes which are set to specified values when the rule is fired and thus provide the forward chaining capability of TIE1.

When TIE1 is invoked, the user specifies the KB to be used and the goal to be sought. TIE1 then reads and parses (via **get_tok**) the specified KB and dynamically allocates the data structures which represent each frame. After each frame is allocated, it is filled with the attribute and rule information specified in the KB and then placed in a hash table to allow quick look up by frame name. Finally, TIE1 considers the goal frame and starts the search for its value by testing each rule in this frame. In the simplified version of TIE1, the name of the first true rule is returned as the value of the goal being sought.

Because TIE1 uses MEX -style user interfaces for the primitive attributes, its frame data structures utilize a list of mex structures with their respective values to be sought:

```
struct alist {
  struct mex *ma;
  char *value;
  struct alist *next;
};
```

A rule list structure is also used and contains the name of the rule, a flag which is used during rule testing and an associated list of attribute-relation-value triplets:

```
struct rlist {
  char *name;
  int flag;
  struct list *triplet;
  struct rlist *next;
};
```

In addition to the attribute list and the rule list, each TIE1 frame structure also contains the name of the frame, its value (when known) and a default value:

```
struct tie {
  char *name;
  char *value;
  char *default;
  struct alist *alist;
  struct rlist *rlist;
};
```

A simplified TIE1 search algorithm, implemented in module "infer", which uses the TIE1 frame data structure is given below and described in the following paragraphs:

```
infer(tieobj)
struct tie *tieobj;
{
struct alist *a;
struct rlist *r;
struct mex *ma;
struct k *known;
int nhypots;
  if(known = get_known(tieobj->name)) {
      tieobj->value = known->value;
      return;
  }
  r = tieobj->rlist;
  for(nhypots=0; r->name; nhypots++, r = r->next)
      r->flag = TRUE;

  for(a=object->alist; a->ma; a = a->next) {
      ma = a->ma;
      if(known = get_known(ma->name))
          a->value = known->value;
      else
      if((newobj = tie_get(ma->name)) != UNKNOWN)
      {
          infer(newobj);
          a->value = newobj->value;
      }
      else
      a->value = user_select(ma);

      put_known(ma->name,a->value);
      nhypots=test_hypots(tieobj,ma->name,nhypots);

      if(nhypots == 0) {
          tieobj->value = tieobj->default;
          break;
      }
  }
  if(nhypots != 0) {
      for(r=tieobj->rlist; r->name; r = r->next)
          if(r->flag == TRUE)
            break;
      tieobj->value = r->name;
  }
  put_known(tieobj->name,tieobj->value);
}
```

Infer is passed the TIE1 frame data structure (**tieobj**) whose name is the goal being sought. Module **get_known** is invoked first to see if the value of that goal (attribute) is already known and if it is, it sets **tieobj's** value to that known value and returns. Otherwise, **tieobj's** rule list is accessed and the values of all the flag slots are set to TRUE. This has the effect of treating all the rules as contending hypotheses which are initially assumed to be true. Then, the frames attribute list (**alist**) is accessed and each attribute's (**a**) value is sought according to the following ordered strategies:

**Look up the attribute's name in the known facts hash table via module get_known and then return the value found there.**

**Look up the attribute's name in the frame hash table via module get_tie and then invoke module infer again (backward chaining) to obtain the value.**

**Ask the user or a data base for the value of the attribute.**

Once a value is obtained for an attribute, its value is put into the facts hash table via module **put_known**. Then module **test_hypots** is invoked to test each rule in light of the new information obtained. Module **test_hypots** sets each rule's flag according to the success or failure of each rule and returns the total number of true rules (hypotheses). If the number of true hypotheses is zero, then the goal value of the frame is set to the default value and the attribute check loop is exited. Otherwise, the search and test strategy is continued for the remaining attributes in the list.

When the attribute search and test loop is exited, a check is made to see if the number of true hypotheses is zero. If this is not the case, then a search is made to find the first true hypothesis and when found this rule's name is assigned to the frame value. Finally, the frame's value is added to the facts hash table.

## HEURISTIC SCHEDULING

The heuristic scheduler developed by the IEPS group is called the Planning And Resource Reasoning (PARR) shell (McLean, 1989). PARR's interactive mode utilizes MEXLIB tools for user interaction and acts like an intelligent assistant to the user. In the batch mode, it simulates the behavior of an expert human scheduler which has heuristics for where activities are to be placed on a timeline. These heuristics include specifications for the priorities, durations and how often the activities are to be scheduled. In addition, the resources, constraints and conflict resolution strategies may be specified. All of these specifications are placed in a KB which describes the way the expert human scheduler would schedule each general activity type (**activity class**).

PARR's architecture is somewhat like a blackboard model (Engelmore, 1988) which builds an activity timeline on a global blackboard and utilizes agents to perform constraint checking, resource management and conflict resolution. When PARR reads the KB, it dynamically allocates internal structures which represent each activity class and fills the slots with the appropriate generic values, thus PARR is also considered a frame based system. PARR's activity class structure is given below:

```
typedef struct {
  int type;
  char *name;
  int priority;
  int repeat;
  long duration;
  int offset;
  int shiftable;
  struct list *resources;
  struct list *constraints;
  struct list *strategies;
  char *subnames;
  struct list *misc_info;
} ACLASS;
```

Given the background of examples discussed so far, most of the members of ACLASS should be self explanatory and have been explained in detail elsewhere (McLean, 1989,1990). An exception is **subnames** which is a string of optional subactivity names.

When PARR creates an instance of an activity class (**ACLASS**) it dynamically allocates a different internal structure which will contain the detailed scheduling information about that particular instance. Among other things, the **EVENT** structure, as it is called, consists of a new structure (**t**) which represent time (**start** and **stop**) and also a pointer to the KB structure (**ACLASS**) which is used to generate the instance. The additional members are **label** and **flag** which are used to store associated information such as orbit numbers, **reslist** which is a resource list and **subacts** which is an array of optional subactivities. The **last** and **next** members are used to link the instances of a given class so that they can be kept in time order.

```
struct t {
  long seconds;
  int date;
};
```

```
typedef struct event {
  ACLASS *ac;
  struct t start;
  struct t stop;
  char *label;
  char flag;
  struct list *reslist;
  struct event *subacts[MAXSUBS+1];
  struct event *last;
  struct event *next;
} EVENT;
```

When an instance (EVENT) is to be created, the information in the activity class is examined so that the start and stop time of the activity can be set. The PARR controller then consults as many as three agents; the constraint checker, the resource manager and the conflict resolver. When invoked, each agent examines the appropriate slot in the activity class structure and performs its specific task. Status messages are then returned after each of the agents has performed its task and the controller makes a decision as to how to proceed with the scheduling of that particular activity. When the activity has passed all its constraint checks and all its resources have been allocated, it is placed on the timeline. The internal representation of this timeline is an array of EVENT structures:

```
EVENT *timeline[MAXCLASSES];
```

The constraint checker uses a rule representation similar to TIE1 (attribute-relation-value triplets) but does not include the implicit backward and forward chaining capabilities because of the simplicity of this type of constraint check. If any rule is violated, a message is constructed which states the constraint rule that was violated and specifies the conflicting value, otherwise a status of OK is returned.

If constraint checking has been passed and resources are required then the resource agent is consulted which, in turn, consults the appropriate resource model. At present, PARR supports a simplified power model and two different types of tape recorder models. If any of the resource models consulted return a status other than OK, a message is built which explains why the resource allocation failed.

If either the constraint checker or the resource allocation agent returns a status other than OK then the conflict resolution agent is consulted. This agent consults the status message and the conflict resolution slot of the activity class and tries to resolve the conflict by either rescheduling the current activity or rescheduling the conflicting activities. To describe the implementation of all of these strategies is beyond the scope of this paper. However, a description of the general approach to conflict resolution may give some insight into how PARR manages conflict resolution by consulting the strategies list and the conflict messages returned from the constraint checker and the resource manager.

## THE CONFLICT RESOLUTION AGENT

The following is a simplified version of the conflict resolution agent which is discribed in the following paragraphs:

```
resolve_conflict(ew)
EVENT *ew;
{
EVENT *rwndo;
struct list *strat;
int status;
int strategy;
char *duration, *newact;

strat = ew->ac->strats;
rwndo = get_resources(ew);
strat = next_strat(strat,&strategy);

  for(status = NOTOK;
      status == NOTOK && startegy != EMPTY;
      strat = next_strat(strat,&strategy)) {

  if(context(strategy,conflict_msg) != OK)
      continue;

  switch(strategy) {
      case START:
          start(&ew,rwndo);
          break;
      case END:
          end(&ew,rwndo);
          break;

      case BEFORE:
          if(before(&ew) == EMPTY)
              continue;
          break;
      case AFTER:
          if(after(&ew) == EMPTY)
            continue;
          break;
      case DELETE:
          if(delete(ew,conflict_msg) == EMPTY)
              continue;
          break;
      case NEXT:
          if(!next(&ew,rwndo))
              continue;
          break;
      case PRIOR:
          if(!prior(&ew,rwndo))
              continue;
          break;
      case DURATION:
          duration = get_duration(strat->line);
          next_time(&ew->stop,ew->start,duration);
          break;
      case BUMP:
          duration = get_duration(strat->line);
          bump_time(&ew->start,duration);
          bump_time(&ew->stop,duration);
          break;
      case ACTIVITY:
          newact = get_newact(strat->line);
          if(activity(ew,newact) == NOTOK)
              continue;
          else
          return(OK);
          break;
      case SHIFT:
          if(shift(ew,conflict_msg) == NOTOK)
              continue;
          break;
      }
      status = do_insert(ew);
    }
    if(status == OK)
        report_success(ew->start,ew->stop);
    return(status);
}
```

**Resolve_conflict** is passed the activity's data structure (ew) that contains its activity class (ac) with the list of conflict resolution strategies (strats). Initially, **get_resource** is invoked to return the event data structure (rwndo) which is the primary resource window (for example, Daylight view) used by this activity. Then, a loop is initialized which processes the strategies list while the status of each try is unsuccessful and strategies remain. In this loop, module **context** is invoked to determine the suitability of the strategy to be tried in view of the conflict message. For example, if the **BEFORE** strategy is to be used and the conflicting activity is a tape dump and the activity to be scheduled uses tape then the strategy may not be suitable because there probably won't be enough tape remaining just before a tape dump. If the context is not suitable then the strategy is skipped.

On the other hand if the strategy is suitable, the appropriate strategy handler is invoked so that the event structure can be modified accordingly. This modification usually includes changing the start and stop times of the activity. If this adjustment is not successful then the strategy is abandoned and control returns to the next strategy. If the strategy is successful then module **do_insert** is invoked with the adjusted start and stop times. **Do_insert** consults the constraint checker and the resource manager again and adds the activity to the timeline if all goes well. The status of **do_insert** is returned and processing continues depending upon its value. If the status is not OK then the next strategy is tried. If the status is OK then the loop is exited, a message is logged and **resolve_conflict** returns the final status.

The following is a brief description of the conflict resolution strategies used by PARR:

### START

Reschedule the activity at the start of a specific resource window by setting the start time of the activity to the start time of the resource window. Alternatively, reschedule the activity at the start of the specified time.

### END

Reschedule the activity at the end of a specific resource window by setting the start time of the activity to the end (stop time) of the resource window.

### BEFORE

Reschedule the activity before the conflicting activity by adjusting the stop time accordingly.

### AFTER

Reschedule the activity to occur after the conflicting activity by adjusting the start time accordingly.

### NEXT, PRIOR

Reschedule the activity in the next or prior resource window by adjusting the start and stop times accordingly.

### DELETE

Delete the conflicting activities. Start and stop times of the current activity are not adjusted. Care is taken not to delete an activity of higher priority or a required resource replenishment.

### DURATION

Shorten the duration of the activity.

### BUMP

Bump the start and stop times by a specified amount (plus or minus) to avoid the conflicting region.

### ACTIVITY

Schedule an alternative activity instead of the current activity type. This strategy temporarily abandons trying to schedule an instance of the current activity class and tries to schedule an instance of another class. When successful, module resolve_conflict returns immediately with a success status. When not successful, the next strategy in the current activity class is tried. Switching to another activity class amounts to a context switch for controlling the behavior of PARR because each activity class contains its own heuristics which are used to create instances of a particular class. Thus, when module **activity** returns, the context of the current activity class (and strategies list) is restored.

### SHIFT

Reschedule the conflicting activities. This strategy also does not change the current activity's start or stop times. When shifting is attempted, the activity class of the conflicting activity is examined to make sure that shifting is allowed. If shifting is allowed then the conflicting activity is temporarily deleted and the start and stop times are adjusted so that it may be rescheduled out of the conflicting range of the current activity. Then module **do_insert_resolve** is invoked with the conflicting activity's adjusted event structure. **Do_insert_resolve**, in turn, checks the constraints and resources for this adjusted activity and also consults with the conflict resolution agent if required. The case may be that more conflicts will occur and that the shifting strategy be applied again to resolve those conflicts. Thus, this type of conflict resolution demonstrates the constraint propagation problems which PARR attempts to solve by use of recursive application of context dependent strategies.

## CONCLUSIONS

The C-based AI technology presented here is not only clearly possible but is in actual use (McLean, 1987). Because this AI technology is part of an integrated set of tools, the experienced software engineer can readily make use of it to build new applications. It is this merging of the AI technology with the standard tools and techniques of experienced software engineers which makes the AI technology so readily usable.

Traditional software development efforts take years to accomplish their goals and start the process by building the system components from scratch. The future requirements for NASA missions will be even more demanding in terms of the number, complexity and configurability of software. In order to solve these problems, software engineers and managers need to get serious about the software reuse issue. This means that not only do the engineers need to be aware of and design for reuse but also that managers allow for a methodology which supports this effort. This methodology includes building systems through reuse of existing software tools, through iterative refinement and prototyping.

The ERBS scheduling system has demonstrated the utility of the software tools approach to maintain an expert planning system (McLean, 1991). This software reuse approach is also being used to develop the Explorer Platform Planning System (EPPS) (McLean, 1990) which will be used by the flight operations team to schedule mission support activities. EPPS is being built by reusing and enhancing the ERBS scheduling system software tools. Although much of the engineering methodology for reuse technology has been defined, the management methodology is lagging and needs further exploration and development.

## ACKNOWLEDGEMENT

The authors wish to thank Patricia Lightfoot at NASA-GSFC/Code 514 and Ellen Stolarik at Bendix Field Engineering Corporation for their support of this work. This work was supported by NASA contracts NAS5-31000 and NAS5-27772.

## REFERENCES

Booch, G. (1991), **Object Oriented Design With Applications**, Benjamin/Cummings.

Durham, R., Reilly, N. B. and Springer, J. B. (1990), “Resource Allocation Planning Helper (RALPH): Lessons Learned,” **Proceedings of the 1990 Goddard Conference on Space Applications of Artificial Intelligence.** Engelmore, R. and Morgan, T. (1988), Blackboard Systems, Addison-Wesley.

Goldberg, A. (1984), **Smalltalk-80: The Interactive Programming Environment**, Addison-Wesley.

Kernighan, B. W. and Ritchie, D. M. (1978), **The C Programming Language**, Prentice-Hall.

Longstreet, D. (1990), “Introduction,” **Software Maintenance and Computers**, IEEE Computer Society Press.

Martin, R. G. (ed.), D. J. Atkinson, M. L. James, D. L. Lawson, H. J. Porta (1990), "Spacecraft Health Automated Reasoning Prototype (SHARP)," **A Report on SHARP and the Voyager Neptune Encounter**, JPL Pulbication.

McLean, D. R. (1986), “The Design And Application Of A Transportable Inference Engine (TIE1),” **Telematics and Informatics**, J. Liebowitz (ed.), Vol 3 No. 3.

McLean, D. R., Littlefield, R. G., and Macoughtry, W. O. (1987), “Defining and Representing Events in a Satellite Scheduling System: the IEPS (Interactive Experimenter Planning System) Approach,” **Proceedings of the 1987 International Telemetering Conference** Vol 23.

McLean, D. R., Littlefield, R. G., and Beyer D. S. (1987), “A Exper Syste fo Schedulin Request fo Communication Link Betwee TDR an ERBS,” **Telematic an Informatics** J. Liebowitz (ed.), Vol 4 No 4.

McLean D R. Yen W L (1989), “PS PARR Pla Specificatio Tool An Plannin An Resourc Reasonin

Shel Fo Us I Satellit Missio Planning," **Proceeding of the 1989 Goddard Conference of Space Applications of Artificial Intelligence.**

McLean, D. R., Page, B. J. Potter, W. J. (1990), "The Explorer Platform Planning System: An Application of a Resource Reasoning Planning Shell," **Proceedings of the First International Symposium on Ground Data Systems for Spacecraft Control.**

McLean, D. R. (1991), "Maintaining An Expert Planning System: A Software Tools Approach." To be published in **Institutionalizing Expert Systems: A Short Handbook for Managers**, J. Liebowitz (ed.).

Minsky, M. (1975), "A Framework for Representing Knowledge," **Psychology of Computer Vision**, P. H. Winston (ed.), McGraw-Hill.

NASA-GSFC (1988), **MEX Portabl Menu-Base Executive.**

NASA-GSFC (1990), **TAE Plus User Interface Developer's Guide.**

Parikh, G. (1986), **Handbook of Software Maintenance**, John Wiley & Sons.

Schildt, H. (1987), **Artificial Intelligence Using C**, McGraw-Hill.

Stroustrup, B. (1986), **The C++ Programming Language**, Addison-Wesley.

Tenant, H. R., Ross, K. M., Saenz, R. M., Thompson, C. W., and Miller, J. R. (1983), "Menu-based Natural Language Understanding," **Proceedings of the Association for Computational Linguistics**, MIT.

Truszkowski, W. (1989), "Prototype Software Reuse Environment at Goddard SFC," **Software Reuse Issues**, Proceedings of a workshop sponsored by NASA Langley Research Center.

Watson, W. (1990), "Introduction," Space Network Control Conference on Resource Allocation Concepts and Approaches Presentations at GSFC, NASA.

# Long Range Science Scheduling for the Hubble Space Telescope

Glenn Miller and Mark Johnston
Space Telescope Science Institute[1]
3700 San Martin Dr.
Baltimore, MD 21218

## Abstract

Observations with NASA's Hubble Space Telescope (HST) are scheduled with the assistance of a long-range scheduling system (Spike) that was developed using artificial intelligence techniques. In earlier papers, we have described the system architecture and the constraint representation and propagation mechanisms. In this paper we describe the development of high-level automated scheduling tools, including tools based on constraint satisfaction techniques and neural networks. The performance of these tools in scheduling HST observations is discussed.

## 1. Introduction

Launched by the Space Shuttle in April 1990, NASA's Hubble Space Telescope (HST) has begun its mission of scientific exploration. Despite a problem with the primary mirror, the HST has already taken data of unprecedented quality for many objects, including Pluto, a gravitational lens, a star cluster and a supernova.

Scheduling the HST is an especially challenging problem since a year's observing program consists of tens of thousands of exposures which are coupled by numerous constraints. Constraints which must be considered include proposer specified constraints (e.g. precedence, timing, restrictions on spacecraft orientation), orbital viewing constraints (e.g. Earth occultations and Sun avoidance), and spacecraft power and communications constraints. The Space Telescope Science Institute (STScI) is responsible for conducting the science operations of the HST, including planning and scheduling observations.

A detailed description of the HST planning and scheduling problem, including a discussion of the individual scheduling constraints is given in Miller, et al. (1987). In a subsequent paper (Miller, et al. 1988), we described the initial development of the Spike planning system including the method of representing constraints and propagating scheduling decisions. This paper continues the series by describing high-level automated scheduling tools and the operational experience of using these tools to produce the science schedules for the HST. Section 2 describes the Spike system and its role in the HST planning and scheduling process. Section 3 discusses the automated scheduling tools. The experience of using these tools for HST testing and operations is given in Section 4.

[1] Operated by the Association of Universities for Research in Astronomy for the National Aeronautics and Space Administration

## 2. Overview of the Spike System

An astronomer wishing to observe with the HST submits a scientific observing proposal. Based on the recommendations of a peer review committee, the Director of the STScI selects which proposals are awarded observing time (refer to Miller, et al. 1987 for a description of the proposal selection process). Proposals are assigned either high or supplemental priority. Unless prevented by unforeseen technical problems, all high priority observations will be executed and constitute about 70% of the estimated available observing time. The supplemental proposals form a pool used to fill out the remainder of the schedule and the choice of a particular supplemental proposal is likely to be based on scheduling and operational considerations. The supplemental pool oversubscribes the available time, so there is only a moderate chance that a particular supplemental program will actually be executed.

The scheduling process begins with the submission of approved observing proposals (see Figure 1). Astronomers submit machine-readable proposals to the STScI using the Remote Proposal Submission System (Jackson, et al. 1987). A year's scheduling pool of about 300 proposals comprises tens of thousands of exposures on a few thousand targets.

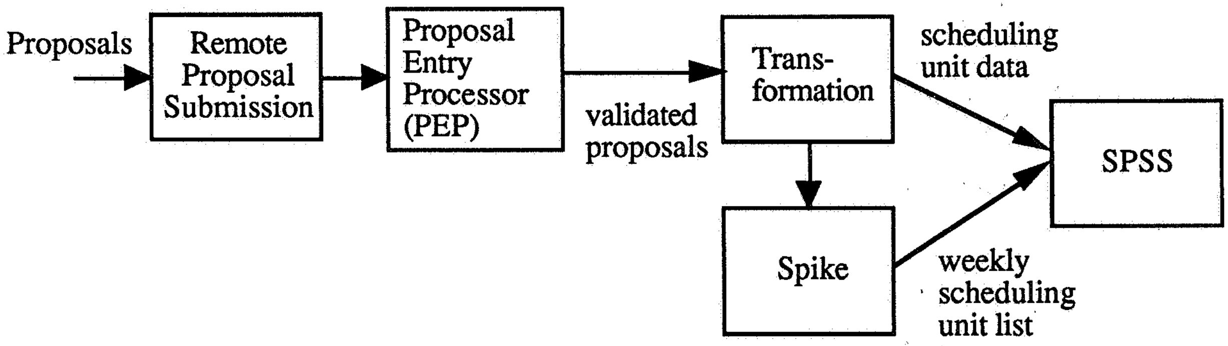

Proposals
Remote Proposal Submission
Proposal Entry Processor (PEP)
validated proposals
Trans-formation
scheduling unit data
SPSS
Spike
weekly scheduling unit list

Figure 1 - Overview of the HST Proposal Processing, Planning and Scheduling Systems. The Remote Proposal Submission System provides proposal preparation tools and accepts proposals via computer. The Proposal Entry Processor (PEP) handles the proposal data. Transformation is used to populate the SPSS system with data. Spike produces a long term plan which is periodically sent to SPSS for short-term scheduling. SPSS returns the results of short-term scheduling to Spike so that the long-term plan can be kept current.

A proposal includes target specifications (position, brightness, etc.) and a list of exposures (target, instrument, operating mode, exposure time, etc.). In order to express scientific constraints on the exposures, a proposal can specify a wide range of properties and interrelationships. For example, exposures may be designated as acquisition or calibration exposures. Some exposures must be executed at particular times or at specific spacecraft roll angles. Ordering and grouping of exposures may be specified as well, and these links may couple exposures separated by many weeks or months. Exposures requiring low background light conditions are identified for execution when HST is in the Earth's shadow. To provide flexibility and a choice of alternative observation plans, exposures may be marked as conditional on some external event (e.g. a ground-based observation) or conditional on the results of other exposures in the proposal. The proposer notifies the STScI when the conditions have been satisfied.

At the STScI, proposal information is contained in the Proposal Entry Processor (PEP) System (Jackson, et al. 1988), which provides tools for entry, editing, evaluation and selection of proposals. If questions or problems are encountered, STScI staff assist the proposer in modifying the proposal.

In order to ensure that the requirements of the proposers and the overall goals of the HST observatory are met, it is necessary to develop a long-range plan. Key considerations for this plan include:

- plan must cover a long time interval (multi-year)
- planning is far in advance of execution, and many constraints can not be predicted in detail in advance
- plan must incorporate a large number of exposures (tens of thousands)
- constraints can couple exposures separated by long time intervals (months to years)
- replanning will be required

A hierarchical approach to scheduling was developed, with the Spike system performing long-range scheduling, and the Science Planning and Scheduling System (SPSS, developed by TRW) performing short-range scheduling. Spike is used to generate multi-year schedules with observations allocated to week-long segments. SPSS does detailed scheduling of observations within a weekly segment.

To be scheduled, the proposal must be cast into a form which can be understood by the SPSS. A PEP proposal expresses the astronomical observations in terms familiar to an astronomer, while the input to SPSS must conform to the design of SPSS, HST and fundamental orbital considerations. This process is called Transformation and is performed by an expert system described by Gerb (1991, see also Rosenthal, et al. 1986). In SPSS, the fundamental data structure is called the **scheduling unit**. Transformation assigns PEP exposures to scheduling units. Transformation performs more than a simple translation of the proposal from PEP to SPSS formats. Transformation chooses implementation scenarios based on the proposer's requirements, orbital conditions and spacecraft constraints.

The Spike system performs long-range scheduling, using Transformation results as input. Spike provides several important features for HST scheduling:

- Constraint representation and propagation mechanism which includes the ability to express human value judgments as well as constraints which can never be violated
- Proposal evaluation tools which allow planners to display observations and constraints on a high-resolution graphics workstation
- Automated and manual scheduling tools
- Window-oriented interface to Spike commands
- Automated tools to track information sent to and received from SPSS

Details of Spike are given in Miller, et al. (1988), Johnston (1990), and Miller and Johnston (1990). We summarize some of the most important features of the Spike system below.

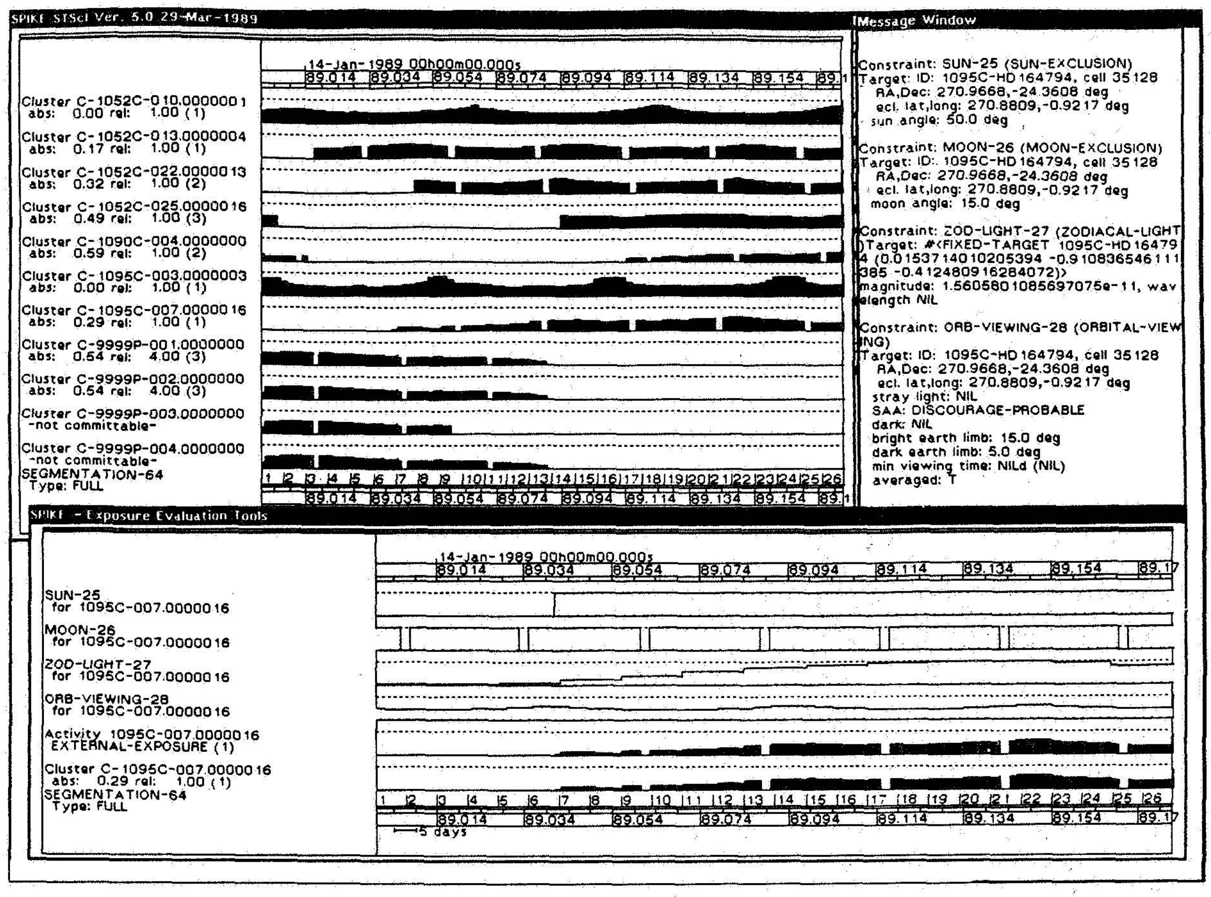
A scheduling **cluster** is the lowest level entity which can be scheduled in Spike. Clusters consist of one or more activities. For the evaluation of single proposals, clusters correspond to scheduling units and the activities are the individual PEP exposures in the scheduling unit. Using graphic tools, a planner can click on a cluster (scheduling unit) and examine its constituent exposures. For scheduling many proposals, the PEP exposure information is discarded and each cluster consists of a single activity which is a scheduling unit. On average, a scheduling unit consists of about 5 PEP exposures. This results in a compression of the amount of activities considered by Spike and a resultant increase in performance.

A **constraint** is any factor that describes when it is possible or desirable to plan an activity. This includes strict constraints ("never point the HST closer than $40^{o}$ to the Sun") and preference constraints ("an deviation of spacecraft roll of up to $30^{o}$ is sometimes allowable but should be avoided unless necessary"). **Absolute constraints** limit the opportunities for a particular activity and are independent of when other activities are planned (e.g. Sun avoidance, guide star availability). **Relative constraints** describes the relationship between two or more activities and change as activities are fixed in the scheduling process (e.g. if activity B must follow activity A by 90 days or more, then fixing a time for A's execution affects B's allowed times). Constraints are represented in Spike by a **suitability** function. This gives the desirability of starting an activity at a particular time. A suitability of 0 means that a start time is forbidden, while a positive suitability indicates that the activity can begin at that time. A suitability of 1 is defined as the nominal suitability, while suitabilities < 1 (but > 0) indicate less desirable, but allowed, start times. This method of constraint representation allows Spike to represent all constraints in a consistent manner and to provide an efficient means for propagating constraints and the effects of scheduling decisions.

In Spike, a long range plan is called a **planning session**. Within a planning session, time is divided into intervals called **segments.** Segments are simply a convenient means to discretize time, creating time "bins". For long-term planning, this allows a significant reduction in the time dimension of the problem without being artificially limiting. A scheduling cluster can be **committed** to a segment, that is, restricted to start during the time interval of the segment. For HST scheduling, the planning session is several years long and there are typically 52 1 week segments per year. The commitments in these segments are periodically transmitted to SPSS for short-term scheduling and subsequent execution. A planning session can have one or more **resource constraints.** These express limitations on resources consumed by clusters committed to a segment, e.g. total exposure time, data volume and real time interaction.

A Spike user can manually commit scheduling units to segments or invoke automated scheduling tools. Figure 2 shows the proposal evaluation tools display. Long term scheduling is complete when all high priority scheduling units have been allocated to weekly segments and supplemental scheduling units have been allocated to oversubscribe the resources to about 20% above capacity.

Several months prior to execution, the list of scheduling units for a week are transmitted to SPSS. After short term scheduling, SPSS returns to Spike the scheduling status of each scheduling unit, i.e. whether or not it was scheduled, the start time of the observation, and the spacecraft roll angle. Generally, all high priority observations will be scheduled and only supplementals will need to be rescheduled by Spike, however it is expected that some rescheduling of highs will be encountered. After execution of the observations by the HST, data is processed by the Post Observation Data Processing System (PODPS) which provides Spike with the definitive list of observations which were executed. Spike compares this to SPSS-supplied schedule and notes to the user any differences.



```
Cluster C-1052C-010.0000001
 abs:  0.00 rel:   1.00 (1)
Cluster C-1052C-013.0000004
 abs:  0.17 rel:   1.00 (1)
Cluster C-1052C-022.0000013
 abs:  0.32 rel:   1.00 (2)
Cluster C-1052C-025.0000016
 abs:  0.49 rel:   1.00 (3)
Cluster C-1090C-004.0000000
 abs:  0.59 rel:   1.00 (2)
Cluster C-1095C-003.0000003
 abs:  0.00 rel:   1.00 (1)
Cluster C-1095C-007.0000016
 abs:  0.29 rel:   1.00 (1)
Cluster C-9999P-001.0000000
 abs:  0.54 rel:   4.00 (3)
Cluster C-9999P-002.0000000
 abs:  0.54 rel:   4.00 (3)
Cluster C-9999P-003.0000000
 -not committable-
Cluster C-9999P-004.0000000
 -not committable-
SEGMENTATION-64
 Type: FULL
```

```
Constraint: SUN-25 (SUN-EXCLUSION)
Target: ID:  1095C-HD164794, cell 35128
   RA,Dec: 270.9668,-24.3608 deg
   ecl. lat,long: 270.8809,-0.9217 deg
  sun angle: 50.0 deg

Constraint: MOON-26 (MOON-EXCLUSION)
Target: ID:. 1095C-HD164794, cell 35128
   RA,Dec: 270.9668,-24.3608 deg
   ecl. lat,long: 270.8809,-0.9217 deg
  moon angle: 15.0 deg

Constraint: ZOD-LIGHT-27 (ZODIACAL-LIGHT
)Target: #<FIXED-TARGET  1095C-HD16479
4 (0.0153714010205394 -0.910836546111
385 -0.412480916284072)>
magnitude: 1.5605801085697075e-11, wav
elength NIL

Constraint: ORB-VIEWING-28 (ORBITAL-VIEW
ING)
Target: ID:  1095C-HD164794, cell 35128
   RA,Dec: 270.9668,-24.3608 deg
   ecl. lat,long: 270.8809,-0.9217 deg
  stray light: NIL
  SAA: DISCOURAGE-PROBABLE
  dark: NIL
  bright earth limb: 15.0 deg
  dark earth limb: 5.0 deg
  min viewing time: NILd (NIL)
  averaged: T
```

```
SUN-25
 for 1095C-007.0000016
MOON-26
 for 1095C-007.0000016
ZOD-LIGHT-27
 for 1095C-007.0000016
ORB-VIEWING-28
 for 1095C-007.0000016
Activity 1095C-007.0000016
 EXTERNAL-EXPOSURE (1)
Cluster C-1095C-007.0000016
 abs:  0.29 rel:   1.00 (1)
SEGMENTATION-64
 Type: FULL
```

Figure 2 - Example screen from Spike showing the scheduling of a few HST observations. The upper left window represents a six month scheduling interval and displays the combined degree of preference (suitability function) for scheduling a number of exposures (running vertically up the window). The lower window shows an expanded time view of one specific exposure and the constraints that contribute to its scheduling preference. The text window on the right displays descriptive information about the schedule, activities, and constraints. The user interacts with the system by clicking on various active regions and selecting from pop-up menus. For example, clicking on the time scale at the bottom of each window permits zooming in or out in time. The user can create new windows and build new displays dynamically.

## 3. Automated Scheduling

The following sections describe the automated scheduling strategies which have been most frequently used in testing and operations: procedural strategies, neural network scheduler and a scheduler motivated by work on constraint satisfaction problems (CSP). We have prototyped other approaches as well. Miller, et al. (1988) describes a rule-based expert system scheduler, while Sponsler (1989) describes a scheduler which uses genetic algorithms to select from a population of neural network schedules.

In any discussion of scheduling techniques it is important to define how the quality of a schedule is measured. One metric which is readily calculable in Spike is "summed suitability" which is the sum over all clusters of the maximum suitability in allowable segments. Before any commitments are made, the summed suitability gives an upper limit to the best possible schedule. Often it is the case that no schedule can actually be this good since the commitment of a cluster to its best segment can force another cluster to be scheduled at a less than ideal time. After commitments are made, the summed suitability measures how well the schedule comes to satisfying the preferences of each observation. Another scheduling metric is the number of clusters which become unschedulable during the scheduling process.

### 3.1. Procedural Scheduler

The Spike system provides a set of procedural scheduling strategies. These work on the following pattern: From all available clusters and their allowable segments, select one cluster and segment according to some criterion (described below). Commit this cluster to the segment and propagate the effects. Repeat this process until no more clusters can be committed. This is an implementation of the so-called "greedy" algorithm (it is "greedy" since it makes a choice on the basis of what is best for one cluster without regard to how other clusters might be affected). As an example, consider the most suitable cluster-segment strategy: The selection criterion is the average cluster suitability in a segment. Each cluster is queried to find its highest average suitability and the segment in which this occurs. The cluster with the highest suitability of all clusters is committed (with ties broken by taking the first cluster examined). This commitment will usually limit the scheduling choices for the remaining clusters, either by constraints between clusters or by resource constraints. This process is repeated until all clusters are scheduled or all remaining clusters are unschedulable.

Other strategies include the most absolute constrained cluster and the most relative-absolute constrained cluster. In the former, the cluster of choice at each step is the one with the smallest total interval duration of non-zero suitability (i.e. the one with the fewest scheduling choices). In the latter, the cluster selected is the one with the largest number of links to other clusters and (in the event that two or more clusters have the same number of links) the smallest total non-zero suitability interval.

The advantage of these strategies is the speed of execution. The main disadvantage, which is well known, is that such simple strategies can lead to very poor schedules (e.g. a large number of unschedulable clusters and clusters scheduled at times of low suitability).

Spike provides a means to control the clusters examined by these strategies. The user can define a focus set which restricts the attention of the procedural strategies to a subset of all clusters. The focus set can be a user-specified set of clusters, or can be defined dynamically according to certain criteria, e.g. priority, absolute or relative constrainedness.

### 3.2. Neural Network Scheduler

In order to provide a more intelligent and flexible approach to scheduling than the procedural strategies, a scheduler based on neural networks was implemented. Neural nets are motivated by models of how biological nervous systems work and have been applied to a variety of problems including pattern recognition, classification, memory and speech understanding (see Tank and Hopfield 1987 for a general introduction to neural nets).

In Spike, a modified Hopfield neural net is used (Adorf and Johnston 1990). The network can be visualized as a rectangular matrix, where the columns represent segments (time) and the rows represent scheduling clusters (see Figure 3). A particular matrix element (neuron) represents the potential commitment of a particular cluster to a particular segment.

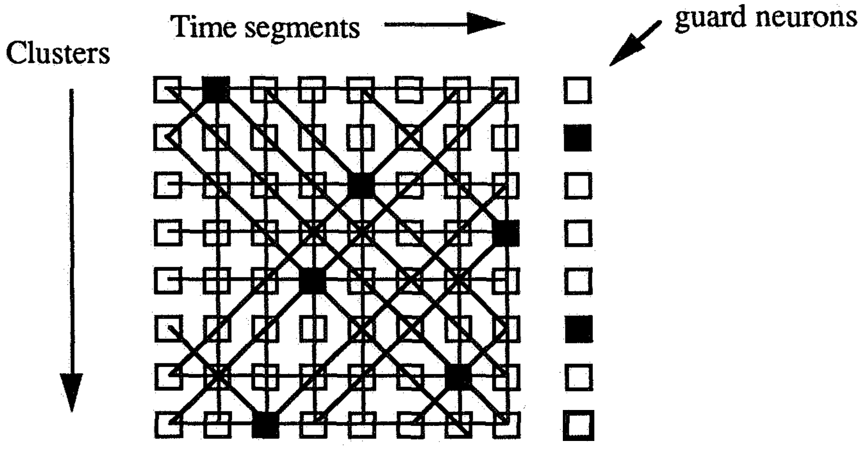


Figure 3 - Neural Network Scheduler.

The output of a neuron is determined by the sum of all inputs. If the sum of the inputs ≤ 0 then the output is zero (the neuron is in the "off" state), otherwise the output is 1 (the neuron is "on" and signifies the commitment of a cluster to a segment). Inputs come from the following:

**absolute suitability:** This term expresses the absolute suitability of a cluster as a function of time. Suitabilities as represented in Spike are combined by multiplication. Inputs in the neural network are combined by addition, so that the weight in the network is simply the logarithm of the average suitability in the segment. This is a constant value and does not change as commitments are made. Zero suitability (or any suitability below a user-specified tolerance) is converted to an inhibitory (negative) weight that cannot be overcome by any amount of positive contributions. This preserves the fundamental constraint representation of Spike suitabilities.

**relative constraints:** For all pairs of clusters A and B which are related, all neurons in A's row are connected to all neurons in B's row. The weight of the connection is determined by the suitability of one cluster given the commitment of the other to a particular segment. As with the previous term, the weight is the logarithm of the suitability, with a large inhibitory value used for suitabilities near zero.

**resource constraints:** When a cluster is committed to a segment, the amount of resources consumed in that segment is updated and any uncommitted clusters which require more than the available resource are given a large inhibitory input.

**guard neuron:** This is discussed below.

The Spike planning session contains the information necessary to determine the absolute and relative suitability terms, thus the neural network does not need to undergo a "training" phase where it "learns" the effect of one cluster on another and adjusts the weight between neurons. (One step of Spike's constraint propagation is to explicitly express all constraints between pairs of clusters. For example, if the proposer specifies a constraint between A and B and between B and C, Spike derives the induced constraint between A and C. We have found that the increase in performance of the scheduling system is well worth the additional computation and computer memory.) The Spike suitability information is saved to a file so that the computation is performed only once and the network can be rapidly initialized from these files.

The network searches for a solution as follows: A row is selected at random and within that row the neuron which is most out of line with its input is selected (i.e. the neuron is on but has an inhibitory total input or the neuron is off but has positive total input). The neuron's state is changed (flipped) to be consistent with the input. The corresponding guard neuron is examined and changed if necessary. This process is repeated until it converges (i.e. the state of all neurons is consistent with their inputs) or a user-defined iteration limit is exceeded.

A Hopfield neural net is guaranteed to converge to a solution, but unfortunately this includes solutions in which not all clusters are committed to segments (in fact a schedule with zero commitments is a solution). The guard neurons are a novel feature of the Spike neural network scheduler and their purpose is to drive the network to committing all clusters. The guard neurons work as follows: For each row (cluster) there is a guard neuron which is connected to all other neurons on that row. When all neurons in the row are off (the cluster is not committed anywhere), the guard neuron is on and applies a positive input to all other neurons on the row, which tends to force them on. When any one neuron in the row is on (the cluster is committed to a segment), the guard neuron applies a large negative input to all other neurons on that row. The addition of the guard neurons, while necessary, breaks the symmetry of the Hopfield model so that convergence is no longer guaranteed. The software monitors the number of steps in the convergence process (i.e. neuron state changes) and halts when a user-specified limit is exceeded. In practice the network runs fast enough that large numbers of trial schedules can be evaluated and compared.

### 3.3. Constraint Satisfaction Formulation

The neural network system described in the previous section was very effective at finding solutions to scheduling and other problems. Investigations into the source of this good performance (Minton, et al. 1990) led to the development of a scheduler which is based on constraint satisfaction problems. The CSP scheduler improved on the performance of the neural network scheduler while allowing a greater control of the scheduling strategies.

A constraint satisfaction problem (CSP) can be described as follows: Consider N variables $v_1$, $v_2$, ..., $v_N$. Each of these variables has a domain of allowed values. The domain is a set of discrete (not continuous) values. There is a set of constraints which limit the allowed values for a variable $v_i$ based on the values of other variables. The problem is to assign a consistent set of values for all variables such that there are no conflicts between constraints and value assignments (i.e. all constraints are satisfied). In the case of HST long-range scheduling, each scheduling unit is represented by a variable and the values allowed for the variables are the segment (week) in which the scheduling unit is to be executed.

The concept of conflicts is central to the CSP scheduler. A conflict occurs when a value for a variable is prohibited by some constraint. A value may have zero conflicts (in which case is consistent with the constraints and current assignments of other variables) or it may have 1 or more conflicts (possibly many more!). The number of conflicts can never be negative. Minton, et al. (1990) found that the key to the neural network's performance was in the way it chose which neuron to update - it chooses the neuron which is most inconsistent with its input and assigns it a value which minimizes the number of other variables that will be in conflict. The CSP scheduler records the number of conflicts on all values of each variable.

The CSP scheduler operates in two distinct phases:

1. Generate an initial guess at a solution (i.e. assign values to all variables). Since this is a guess at a solution, there can be conflicts on the values of some variables.
2. Repair this guess until a solution is found, i.e. until all variables are assigned and none of the assigned values have conflicts.

To be useful, the initial guess must be fairly easy to calculate and must be relatively close to the true solution so that the repair method can find the solution. If it is too computationally intensive to generate the guess, the perhaps some other search technique should be used. If the initial guess is too far from a solution, then the repair method may not find a solution quickly. A useful initial guess method that has been used is to select a variable at random and assign it a value which will result in the minimum number of conflicts with other variable values. A typical repair method selects a variable with the largest number of conflicts on its assigned value and reassigns it a value which has the minimum number of conflicts.

## 4. Experience

This section describes the results of using the Spike system to support the science operations of the HST. Section 4.1 concentrates on the experience gained from scheduling, while Section 4.2 discusses the software engineering aspects of this project.

### 4.1. Planning and Scheduling

HST science operations is divided into cycles in which proposals are solicited from the astronomical community, selected, scheduled and executed. In the long term, cycles will consist of about 1 year of HST observing. Early HST operations consist of two special phases: "Orbital Verification" (OV), which assessed the basic capabilities of the telescope and instruments and "Cycle 0" observations which contain a mix of Science Verification (SV) and Guaranteed Time Observer (GTO) observations. OV ended in November 1990 and Cycle 0 is planned to end in the summer of 1991.

The Spike system was first used to support HST scheduling for Cycle 0. The timeline for SV observations was established by NASA. The STScI Science Planning Branch used Spike to verify this timeline and to schedule GTO observations during weeks when time was available. Spike (and Transformation) was able to uncover a number of scheduling problems with proposals. Often these problems were due to an inadvertent specification by the proposer of inconsistent requirements in the PEP proposal. Although the PEP system performs syntactic checking on proposal information, Spike and Transformation are the first systems which can detect problems related to planning and scheduling. (In particular, accurate instrument overhead times and orbital viewing conditions are calculated by Transformation and Spike and these can reveal problems with the proposal.) We are currently investigating how to incorporate such checks in the PEP Remote Proposal Submission System software which is used by proposers to prepare the proposals. Not only will this provide proposers with immediate feedback of certain classes of problems, but it will lessen the number of delays in the scheduling process due to proposal changes.

Scheduling of the GTO proposals in Cycle 0 was mostly manual. Planners would display individual proposals on the timeline displays and choose times of high suitability for observations. The various procedural commitment strategies were also used in conjunction manual commitments.

The neural network and CSP schedulers will be used for Cycle 1 scheduling. Currently they have been used only in system tests. These tests have been been extensive in that they have included nearly all of the existing proposal pool. Running on a TI Explorer, a multi-year schedule with 1000 scheduling units (corresponding to about 5000 PEP exposures) can be created in less than one hour.

## 4.2. Software Engineering

The Spike system (as well as Transformation) has been developed using "artificial intelligence" techniques and is implemented in Common Lisp. The Lisp environment provides powerful tools to the developer and is clearly capable of supporting large, computation-intensive applications such as HST scheduling.

Careful attention has been paid to the Common Lisp standard and portability. This has allowed some or all of Spike to be run on the TI Explorer, TI MicroExplorer, Sun, Macintosh, Symbolics, and Vax. No source code changes are required to run on different machines, and machine-specific compiler directives are rare (mostly in code accessing the file system). A practical and important benefit of portability has been the ability to incorporate new hardware into our system. Initial development was on the Explorer and MicroExplorer, but within the last year we have added Sun Sparcstations (running Allegro Common Lisp from Franz, Inc.) to development and operations. The Allegro Lisp environment, though not as mature as the TI workstations, is acceptable and the performance of the Sparcstation is impressive. Currently there are 9 Lisp workstations used to support operations, testing and development. A Sun 3 is used as a file server and gateway to other networks. Communications between machines and file access is via TCP/IP for Explorers and NFS for the Sparcstations. We implemented a batch processing utility which allows individual workstations to operate from a central queue. Not only does this allow for efficient use of the operational workstations, but it allows for the development workstations to be pressed into service at times of peak demand.

Common Lisp and the Common Lisp Object System (CLOS) have standardized major portions of the Lisp environment, but there has not yet emerged a high-level standard for windowing systems (X-windows is a standard, but is at a low level). This is a serious obstacle to the development of portable code. This has been recognized by the Lisp community and a number of standards are competing for acceptance.

Parts of the Spike system were prototyped in expert system shells, including KEE (from IntelliCorp) and ART (from Inference), but the current system only uses IntelliCorp's Common Windows subsystem. Why the expert system shells are not used more seems to be the result of two factors. First, Common Lisp now provides some of the features of these shells, in particular, CLOS provides object-oriented programming. Second, Spike does not make use of the paradigms supported by the shells, e.g. forward and backward chaining rules, truth-maintenance and hypothetical reasoning.

Since many requirements of the problem originally ill-defined, a rapid prototyping methodology is an important component of our approach. Users and developers worked very closely on development and operational issues.

## 5. Summary

In this paper we have described the Spike scheduling system which supports long-term science scheduling of the HST. This system is capable of producing long-term schedules consisting of many observations. Three automated schedulers were presented: procedural, neural network and constraint satisfaction based. The latter two schedulers provide a sophisticated and efficient approach to searching for optimal schedules. Lessons learned from HST scheduling and software development were presented and can be applied to other scheduling problems.

We would like to thank the following people for their contributions to the Spike system: Jeff Sponsler, Shon Vick, Anthony Krueger, Michael Lucks, Andrew Gerb, and Mike Rose.

## References

Adorf, H.-M. and Johnston, M. 1990, "A Discrete Stochastic 'Neural Network' Algorithm for Constraint Satisfaction Problems", Proceedings of the International Joint Conference on Neural Networks (IJCNN 90), Piscataway, NJ: IEEE, Vol. III, pp. 917-924.

Gerb, A. 1991, "Transformation Reborn: A New Generation Expert System for Planning HST Operations", Proceedings of the 1991 Goddard Conference on Space Applications of Artificial Intelligence, in press.

Jackson, R., Johnston, M., Miller, G., Lindenmayer, K., Monger, P., Vick, S., Lerner, R. and Richon, J. 1988, "The Proposal Entry Processor: Telescience Applications for Hubble Space Telescope Science Operations", Proceedings of the 1988 Goddard Conference on Space Applications of Artificial Intelligence, ed. J. Rash and P. Hughes, NASA Conference Publication 3009.

Johnston, M., and Miller. G. 1990, "Artificial Intelligence Scheduling for NASA's Hubble Space Telescope", Proceedings of the Fifth Annual Expert Systems in Government Conference,, ed. B. Silverman, V. Huang and S. Post Los Alamitos, IEEE Computer Society Press, pp. 33-39.

Johnston, M. 1990, "SPIKE: AI Scheduling for NASA's Hubble Space Telescope", Proceedings of the Sixth IEEE Conference on Artificial Intelligence Applications" Los Alamitos, CA: IEEE Computer Society Press, pp. 184-190.

Miller, G., Johnston, M., Vick, S., Sponsler, J. and Lindenmayer, K 1988, "Knowledge Based Tools for Hubble Space Telescope Planning and Scheduling: Constraints and Strategies", Proceedings of the 1988 Goddard Conference on Space Applications of Artificial Intelligence, ed. J. Rash and P. Hughes, NASA Conference Publication 3009, reprinted in Telematics and Informatics, 5, pp. 197-212.

Miller, G., Rosenthal, D., Cohen, W. and Johnston, M. 1987, "Expert Systems Tools for Hubble Space Telescope Observation Scheduling" in Proceedings of the 1987 Goddard Conference on Space Applications of Artificial Intelligence and Robotics, reprinted in Telematics and Informatics, 4, 301-311.

Minton, S., Johnston, M., Philips, A and Laird, P 1190, "Solving Large-Scale Constraint Satisfaction and Scheduling Problems Using a Heuristic Repair Method", Proceedings of the Eighth National Conference on Artificial Intelligence, Menlo Park, CA: AAAI Press, pp. 17-24.

Rosenthal, D., Monger, P., Miller, G. and Johnston, M. 1986, “An Expert System for Hubble Space Telescope Ground Support”, Proceedings of the 1986 Goddard Conference on Space Applications of Artificial Intelligence and Robotics.

Sponsler, J. 1989, "Genetic Algorithms Applied to the Scheduling of the Hubble Space Telescope", Proceedings of the 1989 Goddard Conference on Space Applications of Artificial Intelligence, ed. J. Rash, NASA Conference Publication 3033, reprinted in Telematics and Informatics, 6, pp. 181-190.

Tank, D. and Hopfield, J. 1987 “Collective Computation in Neuronlike Circuits”, Scientific American, December 1987, pp 104-114.

N91-22776

# AI TECHNIQUES FOR A SPACE APPLICATION SCHEDULING PROBLEM

N. Thalman, T. Sparn, L. Jaffres, D. Gablehouse, D. Judd, C. Russell
Laboratory for Atmospheric and Space Physics
University of Colorado
Boulder, Colorado 80309

## ABSTRACT

Scheduling is a very complex optimization problem which can be categorized as an NP-complete problem. NP-complete problems are quite diverse, as are the algorithms used in searching for an optimal solution. In most cases, the best solutions that can be derived for these combinatorial explosive problems are near-optimal solutions. Due to the complexity of the scheduling problem, artificial intelligence (AI) techniques can aid in solving these types of problems. This paper examines some of the factors which make space application scheduling problems difficult and presents a fairly new AI-based technique called tabu search as applied to a real scheduling application. The specific problem is concerned with scheduling solar and stellar observations for the SOLar-STellar Irradiance Comparison Experiment (SOLSTICE) instrument in a constrained environment which produces minimum impact on other instruments and maximizes target observation times. The SOLSTICE instrument will fly on-board the Upper Atmosphere Research Satellite (UARS) in 1991, and a similar instrument will fly on the Earth Observing System (EOS).

## INTRODUCTION

As space applications become more operationally complex, automated scheduling systems are a necessity. Missions such as the Hubble Space Telescope, the Upper Atmosphere Research Satellite, and the Earth Observing System will require some level of automated scheduling in order for activities to proceed and resources to be used in a safe and efficient manner. Increasingly, prototyped and operational scheduling systems using artificial intelligence techniques to aid in search and optimization are proving to be successful. The difficulty is to apply these techniques in an efficient manner when solving scheduling problems.

In space applications which have scientific goals, a scheduling system provides a bridge between the science environment and the resource environment. In the operational context it is important to maximize activities with respect to resources. At the same time, scientific objectives must be fulfilled. Coupling these contexts will allow for the maximizing of activities and scientific return with respect to available resources and the resultant schedule will satisfy science objectives.

Research to find good solutions for the scheduling problem includes studies of classical industrial problems involving tasks with predefined precedences and deadlines (such as the Flow-shop, Job-shop or Travelling Salesman problems). Such problems can be solved by general conventional techniques like Branch-and-Bound, PERT or Critical Path Method (CPM) [Wiest and Levy, 1969] [Kaufmann and Desbazeille, 1969]. However, using conventional techniques to solve resource-based scheduling problems for space applications is not easily accomplished.

Scheduling in the space application environment is similar to the factory scheduling problem because it is concerned with the allocation of a limited amount of resources to specific operations over time. Large numbers of activities, resources, and interacting relationships complicate these problems. Procedures that guarantee an optimum solution to scheduling problems are either NP-complete or NP-hard [Ibaraki and Katoh, 1988], most frequently NP-hard in realistic cases. Combinatorially complex problems such as resource-based constraint scheduling are NP-hard. In theory, NP-hard problems have no polynomial algorithm for finding optimal solutions or the number of possible solutions increases so fast that there are no practical results. In most cases, the best solutions for these problems are near optimal. The difficulty of the problem remains mainly in the large number of constraints which have to be satisfied at the same time.

In recent years, artificial intelligence techniques such as neural networks, simulated annealing and genetic algorithms have been prototyped and demonstrated to aid in the solution of resource-based constraint scheduling problems. Another technique gaining recognition is tabu search. Tabu search is a proven search technique for scheduling problems and offers advantages over other techniques with regard to ease of implementation and the flexibility to handle additional considerations not encompassed by the original problem. Some of the applications where tabu search has had great success in terms of performance and adaptability include employee scheduling, space planning and architectural design, travelling salesman problems, and job-shop scheduling problems [Glover, 1990].

The first section of this paper describes factors which make the resource-based constraint scheduling problem difficult. The second section describes the UARS SOLSTICE scheduling problem, followed by a third section which discusses AI-based techniques examined for use on this flight project. The fourth section explains the AI-based tabu search methodology which has been selected for the UARS SOLSTICE scheduling application. And the fifth section discusses the current knowledge-based solution of the UARS SOLSTICE scheduling application and compares it to the tabu search implementation.

## FACTORS IN RESOURCE-BASED CONSTRAINT SCHEDULING

Three predominant factors make the resource-based constraint scheduling problem difficult are constraints, optimization, and search methods. For example, in experiment scheduling, one technique is required to limit the search for all possible experiment schedules and another technique is needed to resolve conflicts that occur in resource assignment for the experiment.

There are essentially two types of constraints in resource-based scheduling: absolute and relative (temporal) constraints. An absolute constraint is an activity or resource that is limited these limitations cannot be violated. For instance, due to the power considerations of the spacecraft, an instrument may only operate during spacecraft sunlight or an instrument may not observe a star when it is occulted by the Earth. A relative constraint describes the relation or relative position between two entities. An example of a relative constraint is when experiment A must be performed before experiment B or experiment A can occur only during experiment C. Collectively, absolute and relative constraints serve to delimit the space of suitable schedules that are generated. However, due to the large number and different types of interacting constraints which can occur, this alone makes conflict resolution of constraints a dynamic and difficult problem. Thus, it is important to represent explicitly and understand constraints to effectively resolve conflicts during the generation of partial schedules.

Since the scientific or operational context will vary from application to application, scheduling goals will also be different. These goals drive the criteria and selection of methods used to schedule activities and resources. Multiple criteria may require several different techniques to generate an optimal schedule. There may need to be compromises in determining whether a schedule is optimal when multiple criteria are used. The difficulty lies in determining the evaluation criteria required to judge the best schedule. Also, as the operational or scientific context of the application changes with time, scheduling goals may change, invalidating current optimization criteria. Therefore it is important to implement a scheduling system that is flexible and can be easily modified under changing conditions.

Scheduling is searching. At each step in the scheduling process, some decision is made about which activities or resources to schedule and when to schedule them. Because of the large number of possible choices, the effort required to search the space of possible schedules is typically exponential. Effective search requires the early identification of good partial schedules which must be judged for their potential for being beneficial to the overall schedule. This is complicated by the fact that scheduling conflicts may not be detected until many steps into the search. It is desirable then to identify a minimal number of past decisions to resolve the conflict, backup, and fix instead of discarding the schedule.

Constraints, optimization, and search methods inherent to resource-based scheduling do make solutions difficult. Since scheduling problems vary from one application to another and are rarely isomorphic to each other, each application will require different solutions and techniques. A solution depends on the characteristics of a given problem which involve: the type of activities (number, complexity, similarity, fixedness, fragmentability, co-dependencies...), the type of resources, time requirements, and goals of the schedule...[Geoffroy et al., 1990]. Also, when a solution is found it needs to be flexible, because most of the time the scheduling environment is constantly evolving from one state to another.

## THE UARS SOLSTICE SCHEDULING PROBLEM

The scheduling application that is currently being examined is for the SOLSTICE instrument, a solar-stellar observing instrument scheduled for launch aboard the UARS (see Figure 1) in November 1991 and the EOS in 1997. The SOLSTICE resource-based constraint scheduling problem consists of coordinating SOLSTICE experimental activity with UARS spacecraft activity such as spacecraft attitude maneuvers or Tracking Data and Relay Satellite contacts, selecting experiments for solar and stellar observation periods and determining instrument platform slew periods and sharing SSPP (Solar Stellar Pointing Platform) resources with two additional UARS instruments. The SOLSTICE instrument has two prime scientific goals: to measure solar ultra violet irradiance with a long term accuracy better than 1% using a set of selected stars as the calibration source, and to provide daily, high accuracy, solar spectra information for use with the atmospheric observing instruments. The SOLSTICE scheduling system must provide a solution for both long- and short-term scientific constraints.

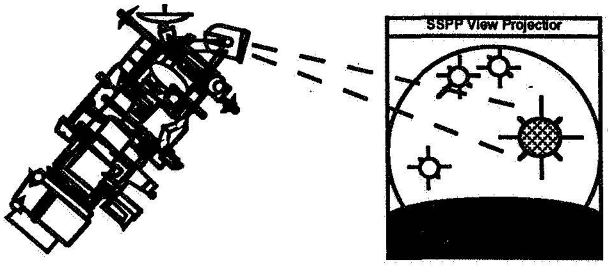


Figure 1 SSPP View from UARS Spacecraft

Scheduling the daily solar spectra activities and coordinating with the other two solar experiments on the SSPP is accomplished during the sunlight portion of the orbits. These activities include calibration of instrument pointing, collection of data for the daily solar spectra and meeting the requirements of the two other solar instruments while remaining within the constraints imposed by the UARS. Because the other two instruments can only view the sun, they place a high priority on the solar viewing resource of the SSPP. Managing the solar viewing resource for all of the instruments is also accomplished by the scheduler.

In scheduling stellar experiments, star observation statistics are used to select the experiments. Star observation statistics are the actual, scheduled and planned experiment data maintained on each star. A set of equations (see Figure 2) using star or target observation statistics is used to derive a star value or experiment priority. In addition, priorities for experiments are calculated based on the accessibility of the star, the success rate at which each star has been observed by the instrument, and an intrinsic star value assigned by the scientific investigator. After a priority is calculated, the experiment is scheduled by priority, star availability, and experiment duration. Stellar experiments are then selected to minimize slew time between stars, thereby optimizing the scientific return. This last step of optimization is to minimize slewing, which is very similar to the travelling salesman problem. It differs only in that the targets are moving with respect to each other over time.

STAR VALUE = TARGET VALUE + ACCESSIBILITY + SUCCESS RATE;

TARGET VALUE = V * (P/S);

where V = initial target value assigned by principal investigator;
S = total number of target observations;
P = number of attempted observations for a given target;

ACCESSIBILITY = K * (D/N);

where K = AC - (RAP - 1);
AC = total number of days for a given target availability period;
RAP = number of remaining days in current target availability period;
D = total number of days in the next unavailability period for a given target;
N = number of days to next unavailable period of given target;

SUCCESS RATE = C * V;

where C = number of actual target observations/total number of attempted target observations;
V = initial target value assigned by principal investigator;

Figure 2 Star Selection Equations

UARS spacecraft events influence the particular stellar or solar experiments scheduled. For instance, a spacecraft yaw around maneuver necessitates that only solar experiments be scheduled during this maneuver. The resulting schedule is an initial science plan with an optimal target acquisition sequence containing scheduled solar and stellar experiments.

## AI TECHNIQUES CONSIDERED FOR THE UARS SOLSTICE APPLICATION

In the application of the scheduling problem for the SOLSTICE, several AI-based techniques were considered. There are several widely known AI-based techniques which have been applied in solving resource-based constraint problems. These were examined and the following search-based techniques were found to be inappropriate for the UARS SOLSTICE scheduling application. These were neural nets, simulated annealing, and genetic algorithms. Neural networks have a reputation of working well in pattern recognition applications, but do not perform well in optimization problems. Simulated annealing and genetic algorithms are assured to produce good results provided appropriate assumptions are made. These two methods are theoretically interesting, but computationally intensive. These methods also select their solutions with some degree of randomness. These two approaches are suitable in certain areas of physics and biology, but not in resource-based scheduling problems for space applications.

Another AI-based search technique named "tabu search" applied well to the UARS SOLSTICE scheduling problem. This search has two significant advantages over the above methods. First, tabu search is easier than the other methods in implementation, and second, it has the advantage of being flexible in handling additional constraints as the scientific or operational requirements change.

## TABU SEARCH METHODOLOGY

Tabu search is a proven method used for solving resource-based optimization problems. This methodology, as illustrated in Figure 3, can be applied to any operation which moves from one state to another. Each move is selected from a set of available moves which create a solution state. This solution state is evaluated to measure its relative value in a local sense. The search provides a guiding framework for generating the best global optimal solution after most classical searches reach a dead end at a local maximum. Tabu search prevents revisiting solutions which have previously been tried and probes for better solutions even after reaching a local maximum. Tabu search does this by using what is called a tabu list. The tabu list contains information regarding past moves or states which could lead to a solution already visited. This list is dynamically updated as the search progresses. If a move under consideration is found to be tabu, criteria are used to decide whether the move is good enough to override the tabu status. The criteria used in making this decision are called aspiration criterion. The aspiration criterion encourage the search for globally superior solutions instead of locally best solutions.

### *Tabu Search In The UARS SOLSTICE Application*

The UARS SOLSTICE scheduling problem has been prototyped using the AI-based tabu search. A description of the algorithm follows. Tabu search starts with an initial schedule or solution state. For this application, the initial schedule is empty. This is also the least desirable schedule. Next, a candidate list of moves is generated. There are three types of moves: inserting, retracting and shifting of an experiment. Only one of these basic moves is applied to the current schedule to obtain a new schedule. For instance, with an empty schedule the candidate list of moves consists of inserting all possible experiments. However, after evaluating each insertion, a new schedule will contain only one experiment or move. The best admissible move among the list of candidate moves is found by evaluating or scoring each move in the candidate list.

An evaluation function scores each move in the candidate list and returns the score. This function is responsible for

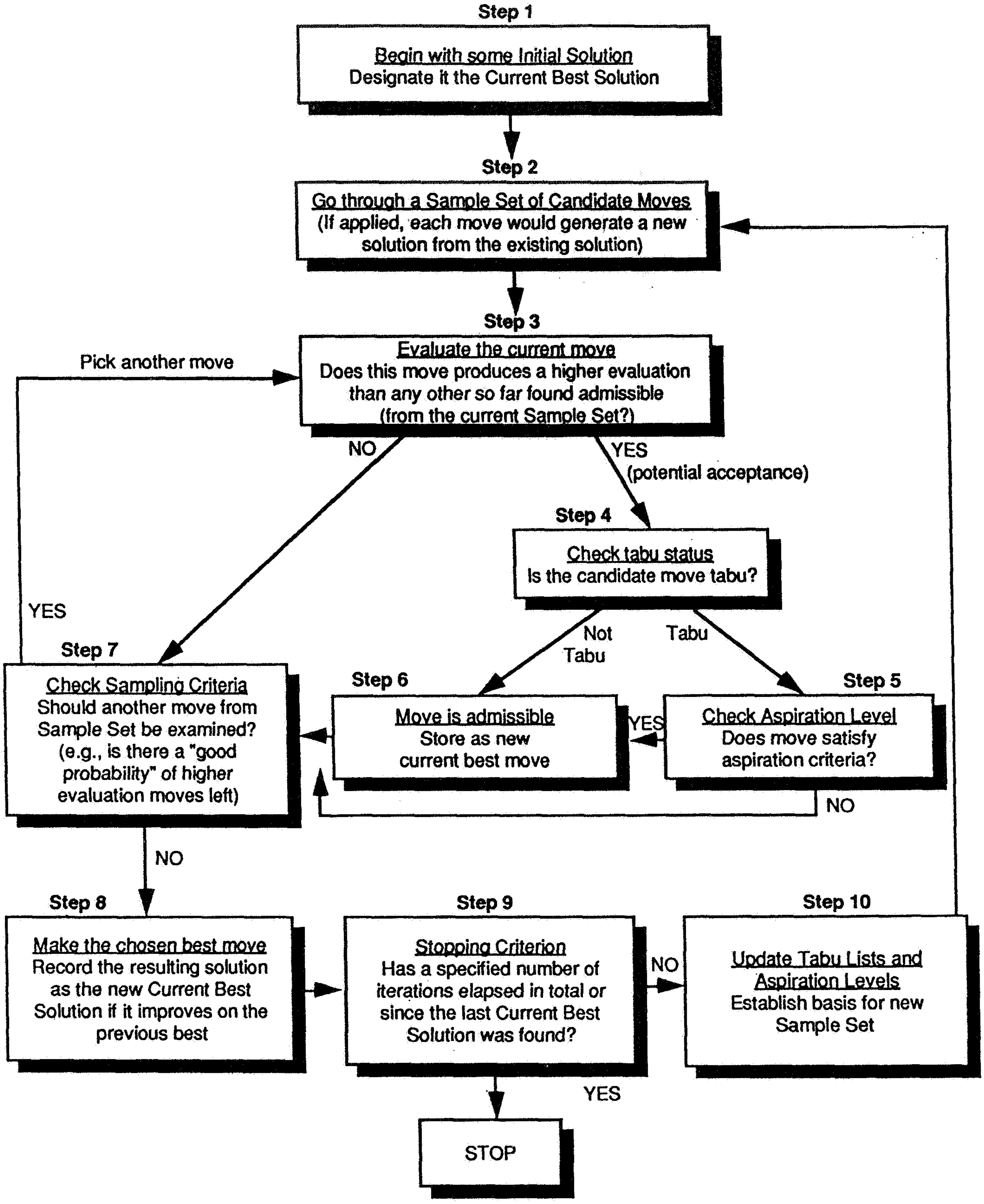

Step 1
Begin with some Initial Solution
Designate it the Current Best Solution
Step 2
Go through a Sample Set of Candidate Moves
(If applied, each move would generate a new solution from the existing solution)
Step 3
Evaluate the current move
Does this move produces a higher evaluation than any other so far found admissible (from the current Sample Set?)
Pick another move
NO
YES
(potential acceptance)
Step 4
Check tabu status
Is the candidate move tabu?
Not Tabu
Tabu
YES
Step 7
Check Sampling Criteria
Should another move from Sample Set be examined? (e.g., is there a "good probability" of higher evaluation moves left)
Step 6
Move is admissible
Store as new current best move
YES
Step 5
Check Aspiration Level
Does move satisfy aspiration criteria?
NO
NO
Step 8
Make the chosen best move
Record the resulting solution as the new Current Best Solution if it improves on the previous best
Step 9
Stopping Criterion
Has a specified number of iterations elapsed in total or since the last Current Best Solution was found?
NO
Step 10
Update Tabu Lists and Aspiration Levels
Establish basis for new Sample Set
YES
STOP

**Figure 3 Tabu Search (used with author's permission: Glover, F., 1990)**

PRECEDING PAGE BLANK NOT FILMED

ensuring that scientific goals and operational constraints are satisfied. Currently, several factors are considered in computing the score. First, the average priority of experiments for each star is considered. It is better to select a higher average priority which is derived from one experiment, than opposed to a lower average priority which is calculated from three experiments. Second, the slew time for targets are compared to the total experiment or observation time for all targets. If total observation time is greater or equal to total slew time then the score is increased. A third factor is scheduling a specified experiment where the experiment duration exceeds the target availability time. A small penalty is assessed because valuable scientific data is lost if the instrument's target is suddenly no longer available and the instrument is viewing deep space. The penalty is small because the experiment may be valuable enough to be considered for scheduling at some other time within the target availability period, thus replacing another experiment of a lesser value. These three factors maximize scientific return by ensuring that priority science is performed and that solar and stellar observing time is maximal. A fourth factor is slew backtracking. This is when a slew from one target to another is performed in the direction opposite to the spacecraft's flight direction. This is an operational constraint of the instrument and a large penalty is assessed to the score when this occurs.

After a move is evaluated, if the score is higher than any found admissible moves, then the next step is to check the tabu list. Checking the tabu list prevents moving to states already visited. If the move does not appear in the tabu list then it becomes the new current best move. However, if the move appears in the tabu list, the move is forbidden, or tabu, and cannot be inserted unless the aspiration criteria is satisfied. To satisfy the aspiration criteria the new value of the move must beat the one in the tabu list. If this is the case the move is permitted and becomes the new current best move. Each move inserted is evaluated until all moves in the list of candidate moves have been evaluated. If none of the moves evaluated is admissible or the schedule cannot be improved, the best schedule found so far is returned and the search is stopped.

If a specified number of iterations has elapsed since the best schedule has been found, then the search is stopped. Otherwise, in order to generate a new list of candidate moves, the tabu list is updated as well as the aspiration level. To update the tabu list, the move opposite the one added becomes tabu, thus resulting in the previous schedule. The aspiration criteria is updated with the new score and becomes the new score to beat. If the move added was previously tabu, then it is deleted from the tabu list. Once all updates are complete the next list of candidate moves is generated.

### *A Knowledge-based Approach Versus Tabu Search*

The UARS SOLSTICE scheduling problem has been prototyped using two AI-based techniques: a knowledge-based approach and tabu search. In this section they are compared and discussed based upon the three factors (constraints, optimization, and search) discussed earlier.

The UARS SOLSTICE scheduling problem has been implemented using a knowledge-based approach. This prototype is operational and is currently being used in the Science User's Resource Expert (SURE), the AI-based scheduling component of the Science Users Resource Planning and Scheduling System (SURPASS). SURPASS is a software tool enabling distributed planning and scheduling and is based on resource allocation and optimization [Thalman and Sparn, 1990]. In this knowledge-based approach, an expert system tool, CLIPS/Ada (C Language Integrated Production Shell/Ada) was used to implement an expert scheduler.

In the knowledge-based approach, constraints were easily represented by rules. Rules facilitate a natural representation of activities, resources, and constraints. CLIPS uses a LISP-like if-then language for rule construction. For example, an instrument operating constraint is to not make stellar observations which are within five degrees of the moon. This constraint easily translates into the English statement, "If a stellar observation is within five degrees of the moon, then do not observe". Rules can be inserted or removed as scientific objectives or mission constraints change. Scheduled activities such as solar or stellar observations were easily represented by templates or the more commonly known structure frames. In the tabu search methodology, constraints are represented in the evaluation function as variables. These variables are assigned a weight which influence the overall score. The difficulty is to derive an equation consisting of the various constraint variables to model a satisfactory solution. This takes time since a great deal of fine-tuning of the evaluation function is required in order to arrive at a practical solution.

In the knowledge-based approach, search and optimization of instrument activity were implemented by using and capturing planning and scheduling heuristics in rules. Since the number of possible experiments or schedules are too large to be able to consider all of them, schedules were generated using rules. These rules reflect the requests of scientists, instrument engineers and operators, and instrument planners and schedulers. Searching which is iterative is not naturally implemented using rules. For example, in selecting a stellar experiment where there were over 300 experiments possible, experiments were selected based on priority and duration. The CLIPS/Ada language provides the capability to implement looping using rules, but performance was somewhat slow. However, since the choice of implementation remains uncertain due to the continuing evolution of strategy and constraints, the rule-based looping might still be legitimate. Search and optimization are inherent to tabu search. These two criteria are the strengths of the algorithm. The tabu search prototype is implemented in Ada. Searching is based on creating a series of moves which consist of randomly selecting experiments and then evaluating the worth of the moves or current schedule. Since the generation of schedules was slow at first, heuristics were added to enhance performance. Each schedule is evaluated in order to arrive at the best global solution. No additional optimization need be built into the algorithm, except for tuning the evaluation function.

*Results*

Results of the tabu prototype have been successful in that it has allowed a detailed understanding of tabu search, the implementation was straight forward, and the results were good solutions. In terms of time, though, the performance of the prototype was slow. In comparing the results of tabu search to the knowledge-based methodology, it took approximately 30 minutes to schedule one hour of stellar observations using tabu search as opposed to five minutes using rules. At the time of this writing, improvements continue to be implemented to increase efficiency as well as improving the results. One such improvement is to start with an initial schedule that contains one experiment per availability period, instead of starting with an empty schedule and inserting an experiment one by one. Since it involves time to insert, performance will most likely improve. Another improvement includes refining the heuristics throughout the code. Refined heuristics are needed to restrict the set of moves or experiments considered in order to improve efficiency. Also, the evaluation function requires more thought such as adjusting the weights of the factors used in the function. A fine-tuned function gives improved scores, thereby reducing the number of moves to be evaluated and increasing performance time.

## CONCLUSION

Similarities were noticed during the development the UARS SOLSTICE scheduling application using different AI methodologies. Our previous experience of knowledge-based systems and our recent experience with tabu search showed us that both methods allowed scheduling heuristics and constraints to be easily added or removed as the problem definition evolved. This is important since most applications are in a state of flux from the initial conception to the operational stage and beyond. Scheduling systems must be flexible to changing conditions and be kept current so that functionality is not lost and the system does not become obsolete.

Traditionally, space scheduling applications have focused on scheduling in the operational context without considering, if applicable, the scientific goals. This concept is dated. Today, AI technology not only allows operational environments to be represented, but scientific objectives are easily represented and incorporated as well. Since the overall idea in scientific-oriented missions is to maximize scientific return, this should be the primary goal. Spacecraft and instrument performance should be maximized to satisfy and support scientific goals.

It is important to prototype realistic scheduling problems. Prototypes implemented using AI techniques provide valuable information in terms of ease of implementation, performance, and correctness of results. However, it is only after these prototypes become operational that we can learn if techniques used are productive and effective throughout the lifetime of the application.

## REFERENCES

1. Geoffroy, A.L., Britt, D.L., Gohring, J.R., The Role of Artificial Intelligence In Scheduling Systems. NASA Conference Publication 3068, 1990 Goddard Conference on Space Applications of Artificial Intelligence, May 1-2, 1990.

2. Glover, F., Tabu Search: A Tutorial. Interfaces 20: July - August, 1990.

3. Ibaraki, T., Katoh, N., Resource Allocation Problems: Algorithmic Approaches. Cambridge, MA, MIT Press.

4. Kaufmann, A., Desbazeille, The Critical Path Method. Gordon and Breach Science Publishers, 1969.

5. Thalman, N.E., Sparn, T.P., Science User Resource Expert (SURE): A Science Planning and Scheduling Assistant For a Resource Based Environment. NASA Conference Publication 3068, 1990 Goddard Conference on Space Applications of Artificial Intelligence, May 1-2, 1990.

6. Wiest, J.D., Levy, F.K., A Management Guide to PERT/CPM. Prentice/Hall, 1969.

# *Diagnosis/Monitoring*

# A Failure Diagnosis and Impact Assessment Prototype for Space Station *Freedom*

Carolyn G. Baker and Christopher A. Marsh

The MITRE Corporation
1120 NASA Road One
Houston, Texas 77058

## Abstract

NASA is investigating the use of advanced automation to enhance crew productivity for Space Station *Freedom* in numerous areas, one being failure management. This paper describes a prototype that diagnoses failure sources and assesses the future impacts of those failures on other *Freedom* entities.

*Keywords*: Failure Diagnosis, Failure Management, Artificial Intelligence

## Introduction

This paper addresses the application of Artificial Intelligence (AI) techniques to failure diagnosis and impact assessment. The Diagnostic Reasoner (DR) is an existing prototype that accepts qualitative status reports that reflect significant changes detected by subordinate systems, determines the likely sources of failures, and projects the impacts of those failures onto other *Freedom* entities. The DR uses AI techniques such as pattern matching, symbolic reasoning, and model-based reasoning to produce a diagnosis to either confirm or correct the subordinate systems' diagnoses.

### Background

The National Aeronautics and Space Administration (NASA) is in the process of designing and implementing a permanently manned space station, Space Station *Freedom*, to be put into low earth orbit. In support of Johnson Space Center's (JSC) Automation and Robotics Division, MITRE is developing a proof-of-concept failure management prototype that demonstrates the use of expert system software to assist the onboard crew via decision support in recovering from failures in a dynamically changing, controlled environment. (Baker, 1990)

Space Station *Freedom* has been defined to have a hierarchical, distributed command and control architecture. The highest level, or Tier I, in the architecture is concerned with global, vehicle-wide issues, and includes automated and manual operations, both on the ground and onboard. Onboard Tier I functions include the failure management function. Tier I entities have knowledge of each of the systems and elements, and how these interact. Tier II entities are systems delineated along functional boundaries, such as the Electrical Power System. Each Tier II entity is controlled by a Tier II manager that has knowledge about its own status, but not of the other Tier II entities. With no direct knowledge of other Tier II entities, each Tier II manager produces and reports dynamic data representing the current status and configuration of its components to Tier I for a global assessment of the enterprise as a whole. (NASA, 1989)

The failure management prototype described comprises two components: the DR diagnoses and analyzes the failure; the Recovery Expert (Rx) formulates sets of possible actions to take (Courses of Action) for recovery. The DR and the Rx act at the Tier I level for failure diagnosis and recovery. This article describes the DR; a companion article (Hammen, 1991) focuses on the Rx. The DR and the Rx are not completely automated functions, but are computerized decision aids in which the user is provided interaction capabilities throughout the failure management process.

We adhere to the significant architecture requirements imposed by *Freedom's* environment, as we hope to influence *Freedom's* design with our approach. In particular, the prototypes rely on the existence of hierarchical, distributed management functions. Constraints placed on the prototypes shape the architecture of the software, but do not limit its usefulness to *Freedom* or to space-related domains. This software can be used in other domains without changing the computational portion of the application, instead changing only the data that describe the environment.

The technologies used in the implementation of the failure management prototype are model-based reasoning (the DR) (Davis, 1984) and goal-directed planning (the Rx). The objective of the prototype is to show how we use these technologies, and how the technologies work together synergistically using the same knowledge representation for both the DR and the Rx. The purpose of developing this prototype is to address a new approach to the problem, to apply advanced technologies to it, to examine all aspects of failure management in one integrated application, and to do this in a general way so that the same technology can be used in other domains. The prototype was developed on VAXstation™ 3100 workstations under VMS™, using ART™/Ada and TAE Plus™.

Figure 1 shows an overview of the environment in which the DR and the Rx prototypes operate. The DR receives Tier II reports of significant, qualitative changes. The DR identifies the suspected causes of the failure, and determines the current and possible future impacts these suspected failures are likely to cause on other *Freedom* entities. Once the DR has diagnosed the problem and determined the immediate and downstream effects and impacts of the failure, the Rx uses this information to formulate solutions, or Courses of Action, for recovery. Each Course of Action deals not only with the immediate problem, but also determines what actions to take to mitigate the constraints imposed by the failure. The Procedures Interpreter, an earlier prototype not described in this paper, executes the pre-defined procedures identified in the selected Course of Action, resulting in commands to the Tier II managers. The effects of these commands on the Tier II systems is, in turn, reported to the DR as changes in status or configuration, thereby closing the loop between Tier I and II.

## DR Overview

The DR is responsible for determining the likely sources of a failure, and projecting the future impacts of the failure on the rest of the station. The DR uses status reports from the Tier II managers along with structural and behavioral data contained in the Component Model to generate lists of suspects that might be responsible for the failure. These Suspect Lists are updated, merged, and deleted to show the current reasoning of the DR. For each suspect identified, the DR

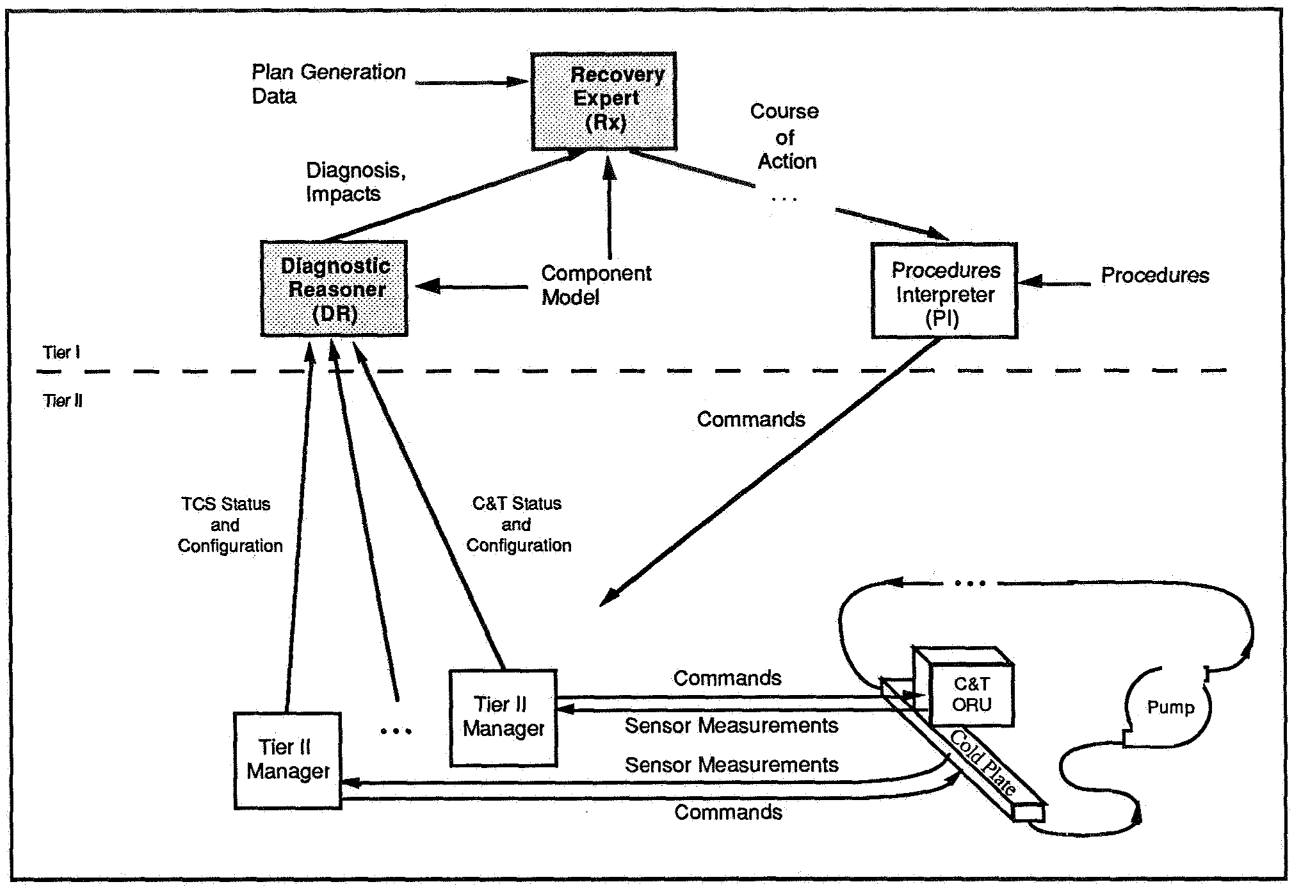


Figure 1. Overview of the DR/Rx Process Flow

uses the Impact Model to assess and project the immediate and future impacts of the failure onto other *Freedom* entities. The suspects and the assessment of their impacts, or Impact Sequences, are passed to the Rx for generation of appropriate Courses of Action for recovery. A high level diagram of the DR processing is shown in Figure 2.

The Tier II managers report significant changes to Tier I. A significant change occurs when the state of some component crosses a qualitative assessment boundary (for example, a component changes from a "nominal" state to a "failed" state, or from a "failed" state to a "nominal" state). In order to determine if there is a significant change, the Tier II manager must assess and evaluate the operational data that it directly monitors. The data that Tier II reports to Tier I represent a qualitative assessment of the system components. The directly-measured quantitative data are also available to Tier I on request.

The DR receives the Tier II reports, and reflects the information in the Component Model, a schematic-like model of the enterprise containing information about both the structure of the enterprise and the expected behavior (and known possible misbehaviors) of the components. If the reports received by the DR indicate a failure, the DR determines possible causes of the failure by searching the Component Model for defined causes of observed misbehaviors. The conclusions drawn by the DR are placed in the Suspect Lists. For each suspect identified, the DR uses the

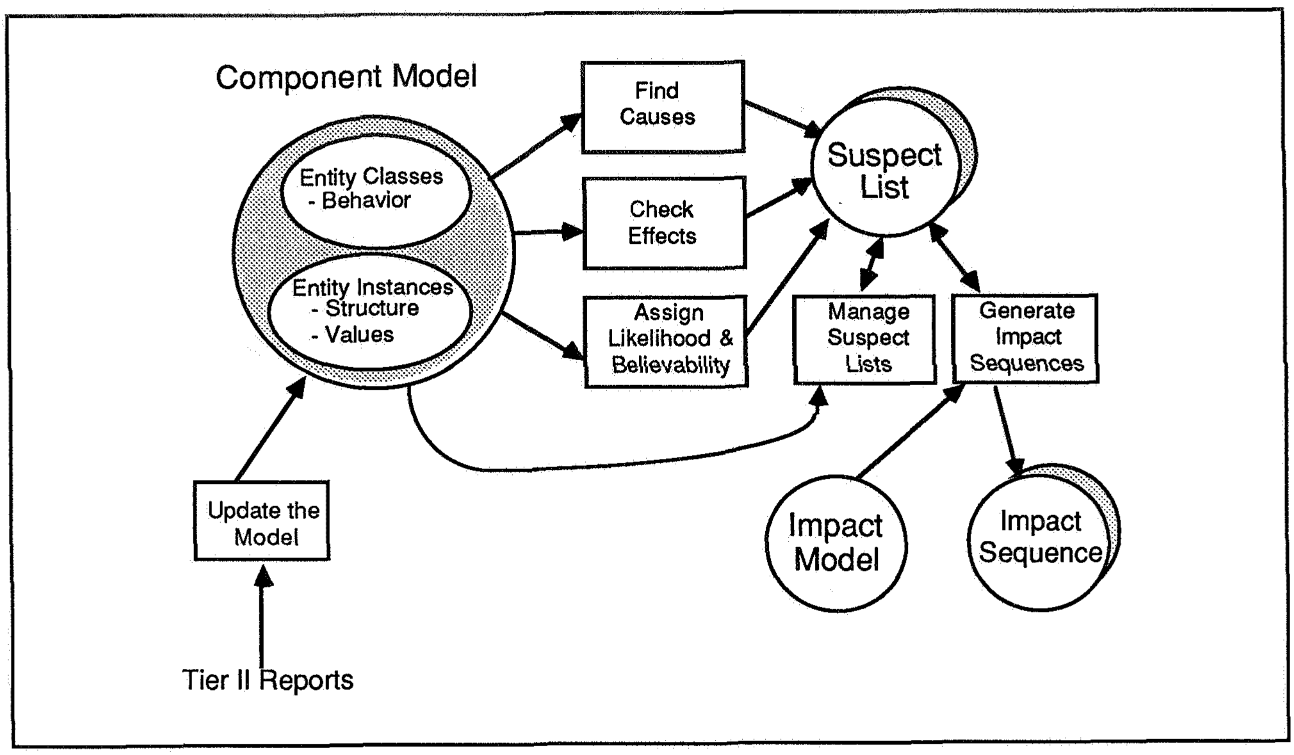


**Figure 2. Diagnostic Reasoner Processing**

Impact Model to determine the immediate and possible subsequent impacts, and expresses them as a cascade of cause and effect in an Impact Sequence.

Once the DR has diagnosed the problem and determined the immediate and downstream effects, the Rx formulates alternative plans for recovery using the Suspect Lists and Impact Sequences.

## Externally Visible Data

The DR either corrects or confirms Tier II diagnoses by updating and using data in the Component Model and the Impact Model. Both these models incorporate an element of time to cope with the dynamic environment. The DR diagnoses, which include both the suspected causes of the failures and the Impact Sequences, are placed in Suspect Lists. We describe the relative time representation and these four data types in this section.

### Relative Time Representation

The prototypes are expected to operate in a dynamically-changing environment, where the state of the vehicle changes with time. Timing and temporal relationships among detected events, projected future events, and planned activities are very important to the functioning of the prototypes. The problem is not how to express the specific time at which detected events have occurred (an absolute representation of time), but rather how to express the relative time frame in which future events are expected to occur: this is a matter of timing rather than time.

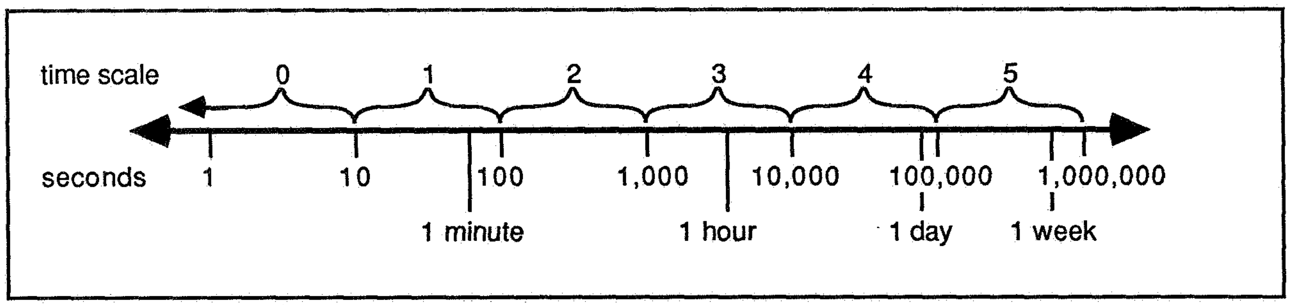


**Figure 3. Logarithmic Representation of Relative Timing.**

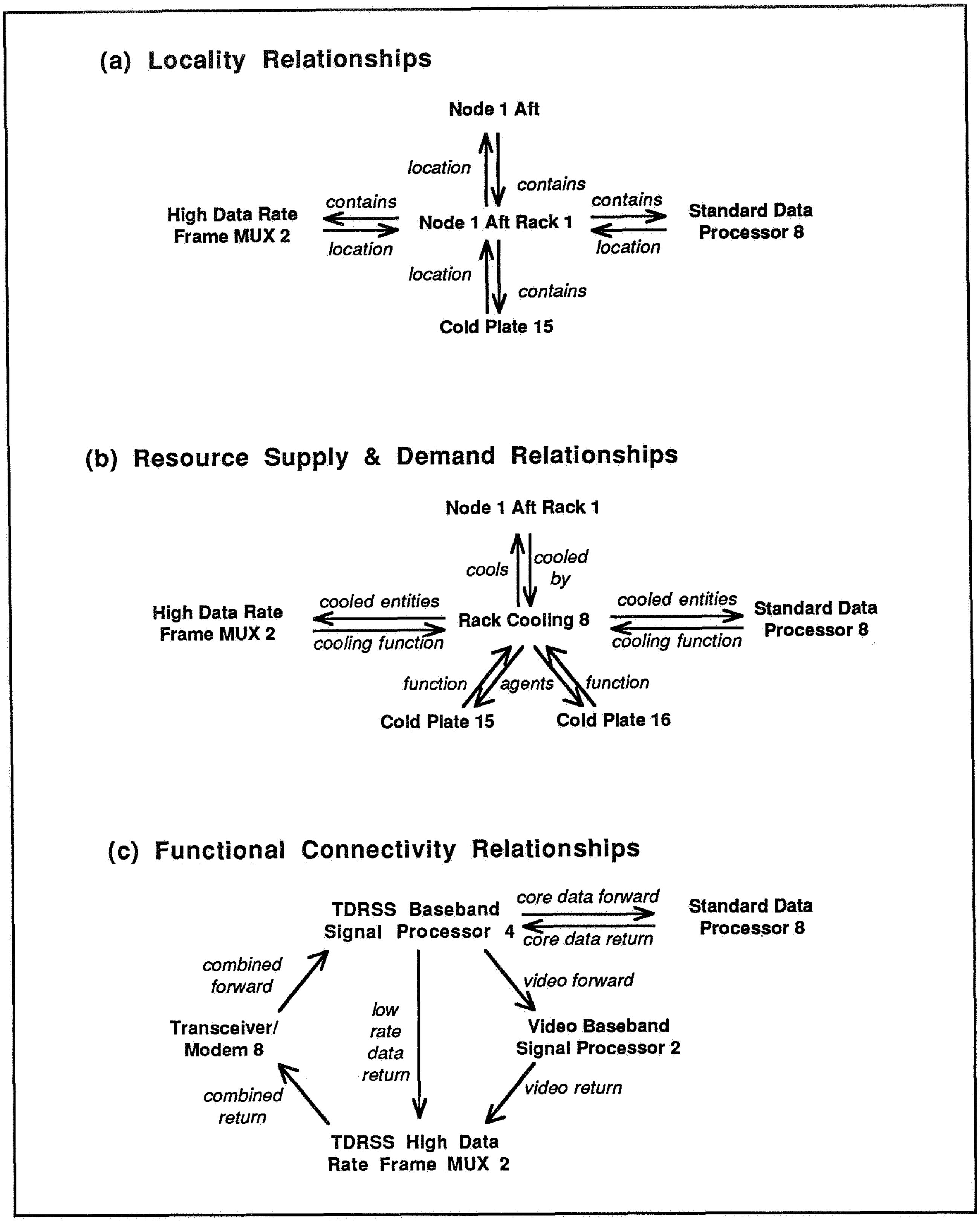
We express relative time on a logarithmic scale of seconds, chunking the time into discrete integral intervals. Figure 3 illustrates this use of time intervals. One example of using this scale is the expression of timing between some event and the expected impact to be felt as a result. If the timing information has a value of 0, the impact is expected to be felt within a maximum of ten seconds. If the timing information has a value of 2, the impact should be felt in 100 to 1000 seconds.

## Component Model

The Component Model, a schematic-like representation of the space station and its components, is at the heart of both the DR and the Rx processing. The Component Model incorporates structural, behavioral, and status information. The structural information identifies each individual component's relationships with other components, both physically and functionally. The behavior information identifies, for each type or class of component, the causes and effects of particular conditions with respect to that component type's health and mode of operation. The behavior information describes both internal causal consequences and behaviors across component boundaries. The status information identifies, for each individual component that is operating, the current operational values related to the component's mode of operation, equipment health, and key operational values that are related to behavior.

Figure 4 illustrates three types of inter-component relationships expressed in the Component Model. Locality relationships give information regarding the position in the space station in which components are physically located. Resource Supply and Demand relationships show the functional inter-component dependencies. Note the links between these representations: for example, Node 1 Aft Rack 1 is defined in the Locality relationships and referred to in the Resource Supply and Demand relationships. Resource Supply and Demand relationships show how individual components are functionally aggregated to provide high-level capabilities. Func-tional Connectivity relationships show how the individual components are physically linked together.

Figure 5 illustrates the hierarchical nature of the systems' representation, using the Thermal Control System (TCS) and Communication and Tracking System (C&T) as examples. This representation also encompasses information about characteristics of the components (shown in more detail in Figure 6). Connections between the Levels (0, 1, 2 and 3) show the inheritance pathway in the Component Model. At the lowest level in the figure, the instantiation of specific components is shown, where each component class is likely to have many individual instantiations.



(a) Locality Relationships

(b) Resource Supply & Demand Relationships

(c) Functional Connectivity Relationships

Figure 4. Relationship Types Within the Component Model

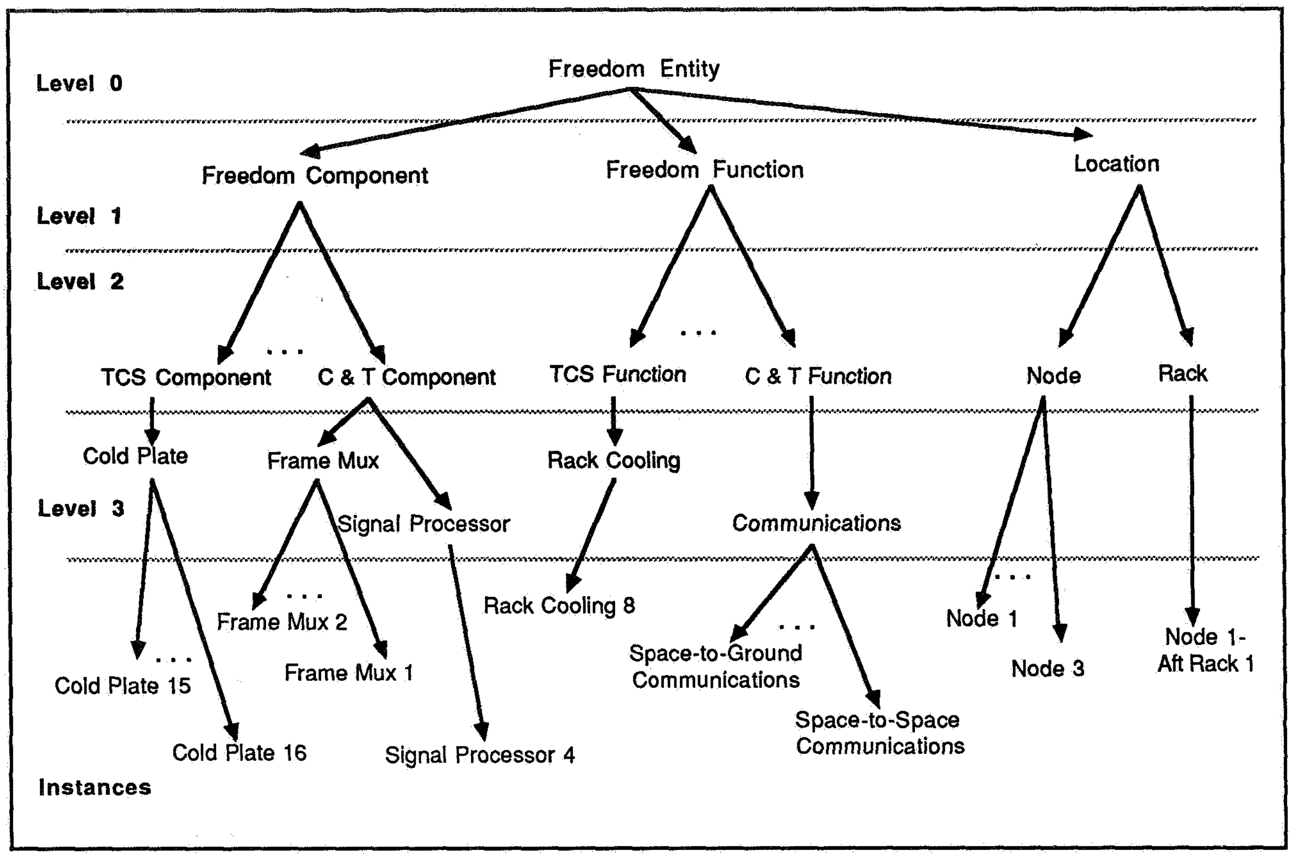


**Figure 5. Component Model Hierarchy and Inheritance Structure.**

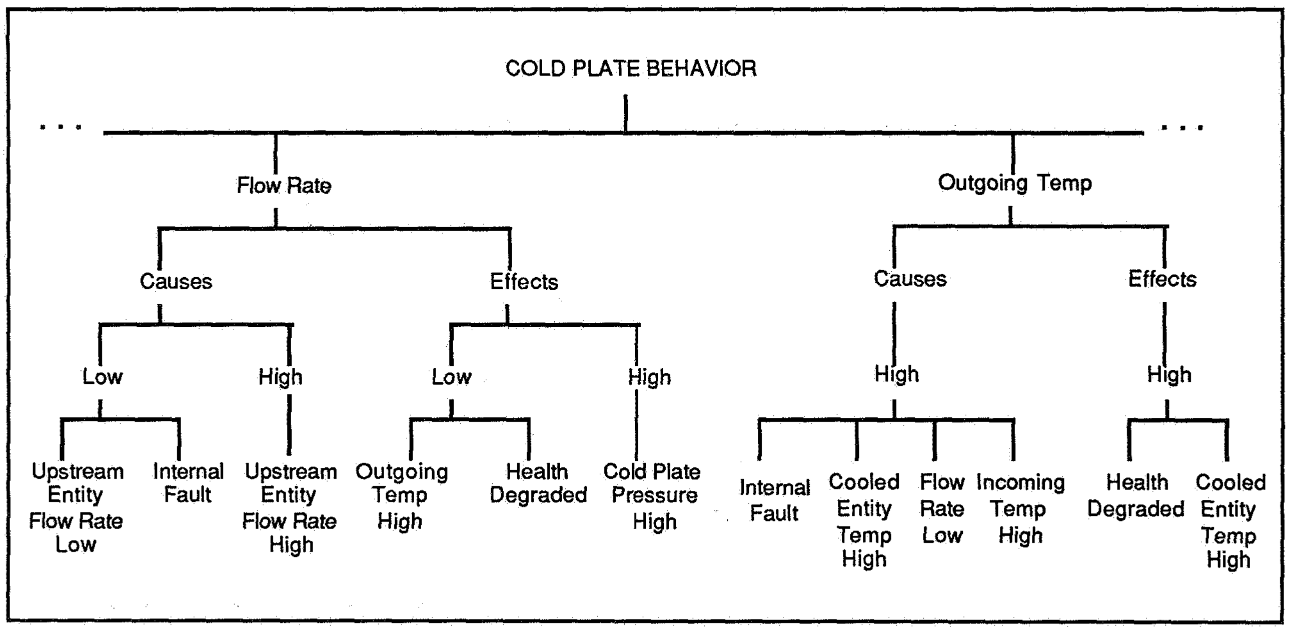


**Figure 6. Behavior Measures for a Cold Plate Component Type.**

The description of a specific component's behavior and characteristics is based on the generic description of a class of related components; the class description, in turn, is defined as a hierarchy of descriptions. The description of a class of components includes descriptions of types of relationships with other components, component behavior causes and effects, and attribute definitions for status information. The values for the status information includes the specific behavior measures and operating conditions of each individual component.

Figure 6 elaborates on a subset of the behavior information for the specific component class Cold Plate. This information is expressed at Level 3 of the hierarchy as shown in Figure 5. Behavior information for a component type is expressed as a cause-and-effect model for possible observed behaviors. In Figure 6, for example, two possible causes for a cold plate to exhibit a low flow rate are "Upstream Entity Flow Rate Low", or "Internal Failure". Upstream Entity is an abstraction that is instantiated for the specific component instances using information extracted from Functional Connectivity relationships (Figure 4(c)). Figure 7 illustrates the dynamic operational values related to measurements of behavior for a component instance. This type of representation is repeated for each component in the Component Model.

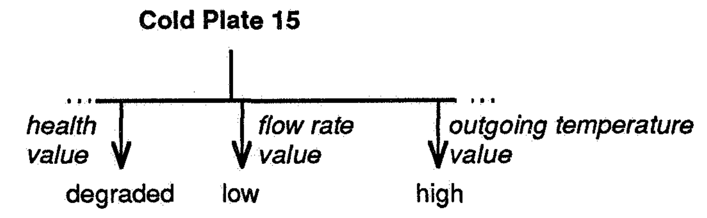


**Figure 7. Cold Plate Instantiation.**

## Suspect List

A Suspect List identifies a set of related observed (off-nominal) behavior attribute values and those components whose off-nominal performance could result in those observations. Figure 8 shows the structure of a Suspect List.

A Suspect List is associated with a specific failure. When diagnosing a failure, the same failure situation might be explained by more than one possible cause; that is, one failure situation can yield more than one possible diagnosis. In this case, the DR generates a single Suspect List containing several Suspect Groups, each group providing a different explanation for the same failure. In some cases, a single component may not be the cause of failure. A set of observable behaviors resulting from multiple component failures may be jointly responsible for the failure. In this event, the DR lists all of the components as a possible cause in a single Suspect Group.

A Suspect List also can identify key unknown behavioral measures: operational behavior measures whose value is not available directly. The assessment of some behavioral measures will require special resources (e.g., asking an astronaut to execute a procedure to collect data) or will induce inter-system interactions (e.g., taking a component offline to perform diagnostics). When the DR encounters one of these unknown measures along one of its diagnostic pathways, it posts the measure in the Suspect List as an unknown whose assessment should help refine the diagnosis.

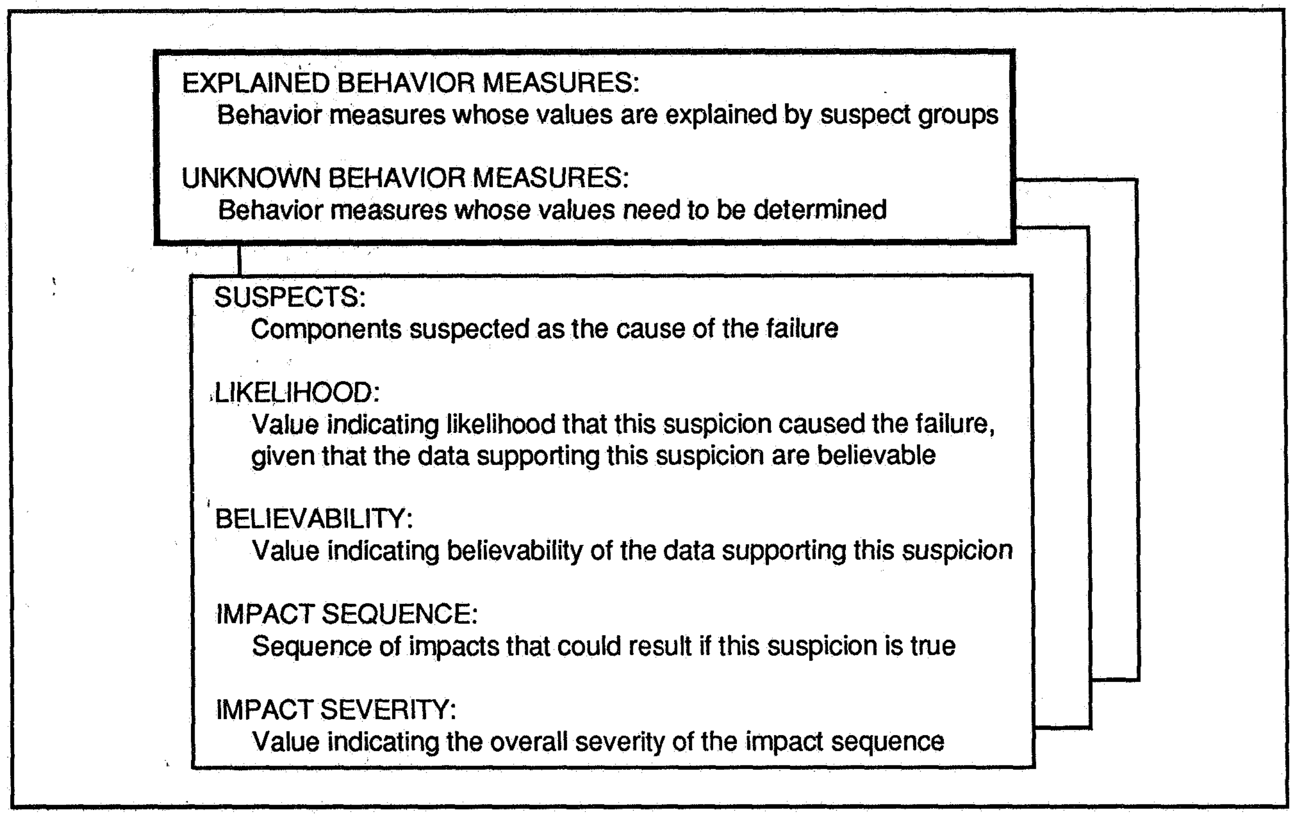

EXPLAINED BEHAVIOR MEASURES:
Behavior measures whose values are explained by suspect groups
UNKNOWN BEHAVIOR MEASURES:
Behavior measures whose values need to be determined
SUSPECTS:
Components suspected as the cause of the failure
LIKELIHOOD:
Value indicating likelihood that this suspicion caused the failure, given that the data supporting this suspicion are believable
BELIEVABILITY:
Value indicating believability of the data supporting this suspicion
IMPACT SEQUENCE:
Sequence of impacts that could result if this suspicion is true
IMPACT SEVERITY:
Value indicating the overall severity of the impact sequence

**Figure 8. Suspect List Structure**

For each Suspect Group in the Suspect List, the DR adds likelihood and believability values, as well as impact severity values and an Impact Sequence. Likelihood indicates how likely it is that the Suspect Group caused the failure: for example, a component that fails frequently is more likely to be the cause than one that fails rarely. Believability indicates how reliable the data are that are used to determine the Suspect Group as the cause of failure: for example, some sensors are known to frequently fail or be unreliable. The impact severity value is an indication of the severity of the failure based on the severity of projected consequences. This value is used by the Rx to determine whether a recovery Course of Action is generated for that suspect for safety's sake, regardless of the likelihood that it really is the source of the failure.

## Impact Model

The Impact Model is used to predict the cascade of effects that are expected to result from a specific failure type, or Impact Sequence. The Impact Model contains information much like that needed by a person to formulate Failure Mode, Effects, and Criticality Analysis (FMECA) trees. This includes, for each system in the enterprise, and for each component in each system, the relationships between each component's failure modes and the effects of those failures with respect to each system operating mode as well as the operational requirements of those modes. These data provide the basis for determining the effect of the failure on the entire enterprise with respect to the time of the failure's occurrence, the component's as well as the system's operating mode at that time, and the particular set of operational requirements against which the effect of the failure must be assessed. (Ireson, 1988)

The Impact Model focuses on the immediate and near-term future operational causes and effects. The data contained in this model is partitioned into sections, where one section expresses one type of cause-and-effect sequence. Each cause-and-effect sequence contains information such as the failure type, the component or function class affected by or involved in the failure, the failure type expected to appear as a result of the failure, the component(s) or function(s) to which the failure type is expected to propagate, a heuristically assigned severity value for the failure type, and the temporal aspects of the failure. For a specific failure, these sequences are instantiated and linked together, based on the current configuration of the enterprise.

Figure 9 provides an illustration of a portion of the Impact Model: the Impaired Operations of the Active Thermal Control System (ATCS) failure type. One cause-and-effect sequence is associated with this failure type (i.e., a Reduced Resource Supply in the ATCS cooling); however, the model structure can support multiple effects. Again, aspects of the model are represented symbolically (e.g., ATCS Cooling-Location), and must be instantiated for specific failures.

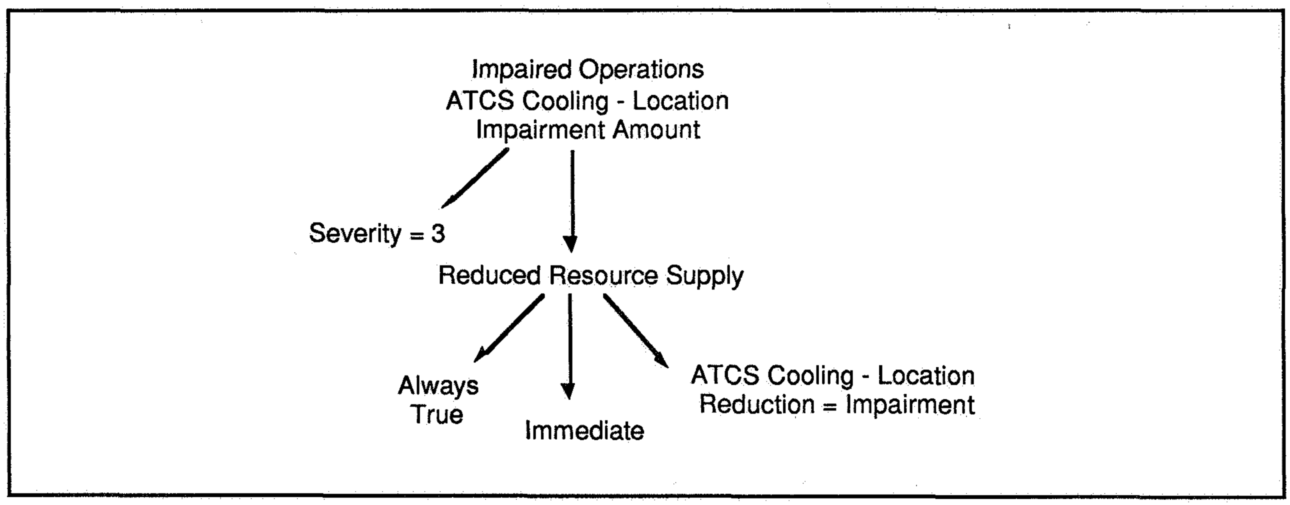


**Figure 9. Impact Model Example**

## Impact Sequence

The Impact Sequence expresses the cascade of immediate and subsequent effects seen as the result of a particular failure. The concept is similar to that of the FMECA, expressed in the following example from the seventeen-century of a component failure that resulted in catastrophic failure: "For want of a nail the shoe was lost, for want of a shoe the horse was lost, for want of a horse the rider was lost ..." and so goes the cascade of subsequent effects of a minor failure in the communication system until, at last, the kingdom was lost. (Ireson, 1988)

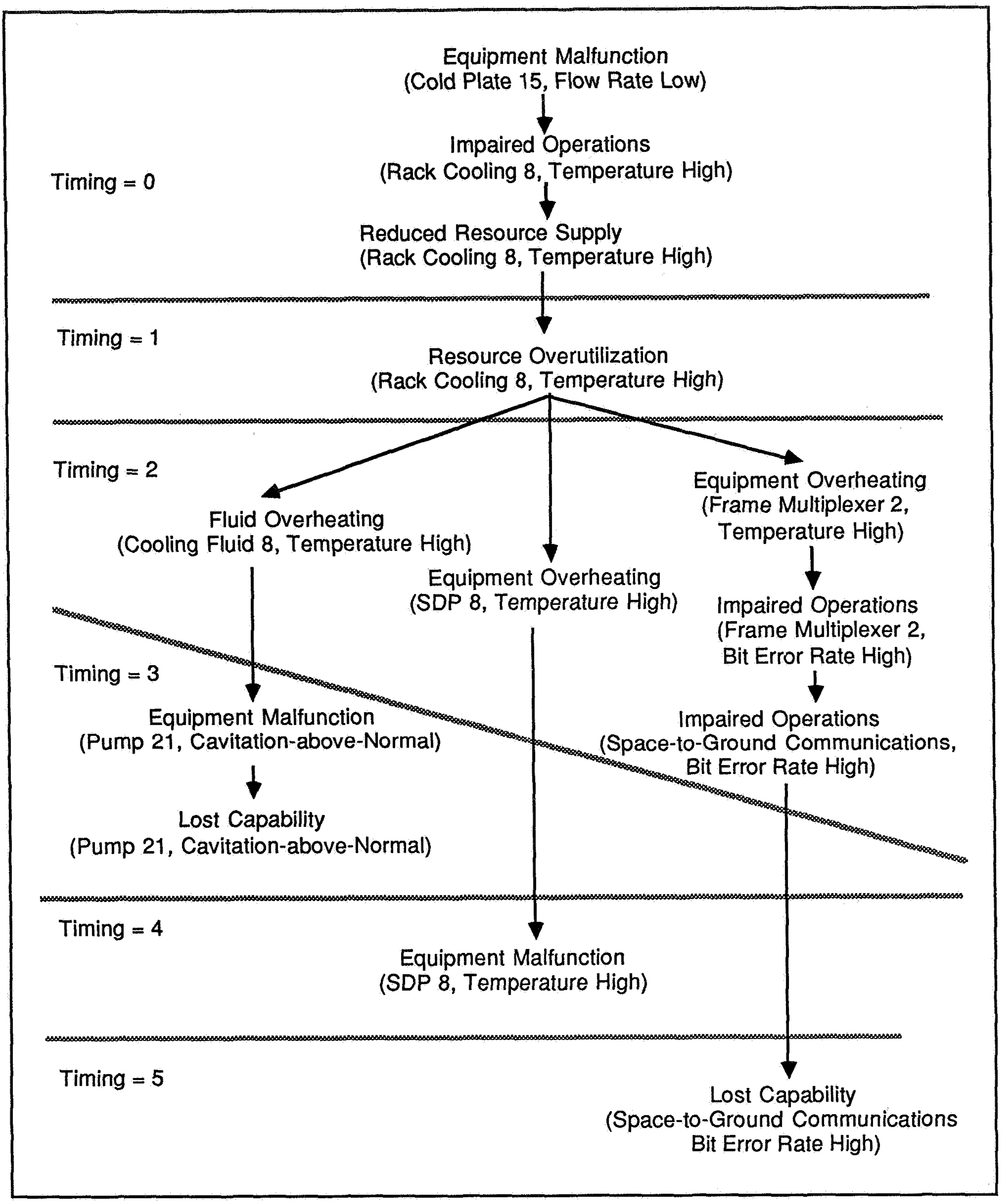


**Figure 10. Impact Sequence**

The Impact Sequence is a tree-like structure constructed by starting with a root cause (e.g., Equipment Malfunction), examining it in the given situation, and linking together the causes and effects as they apply to situation. As with FMECA work, it is not feasible to identify and control every failure in every system with the potential for catastrophic effects; rather, the objective of the Impact Sequence is to provide a sound basis (to the RX) for guiding the allocation of resources such that the probability catastrophic failure is reduced as much as possible. (Ireson, 1988) Figure 10 illustrates an Impact Sequence.

## The DR Processing

The following sections describe the DR processing, as illustrated in Figure 2, using the preceding data descriptions.

### Updating the Component Model

The Tier II managers notify the DR of changes in the both the status and configuration of system components through system reports. Only significant qualitative changes (for example, a cold plate's temperature changing from "nominal" to "hot") are reported to the DR; minor quantitative deviations are not. The DR uses the information in the system reports to update the Component Model.

### Identifying Suspects

When the system reports indicate a problem, the DR searches for suspected causes using the reported status and the modeled behavioral cause-and-effect. Once a suspect is identified, it is placed as a Suspect Group within a Suspect List. If the DR identifies a multiple source failure, the Suspect Group will contain more than one suspect. After a suspect is identified, the DR verifies that the expected behavioral effects have occurred with respect to the suspect(s) and their related components in the expected time frame. This process is repeated until no new suspects can be identified.

For each Suspect Group, a likelihood value, which incorporates two factors, is assigned. The first factor considers how likely the Suspect Group is to be the cause of the failure, taking into account the component's failure rate. The second considers the likelihood that a multi-component Suspect Group is the cause of the failure. A believability value, also assigned to each Suspect Group, is an assessment of how much belief is given to the information from the systems. Suspect Groups containing component known to have sensors that produce unreliable values listed as the cause will have a low believability. The inclusion of a believability value for a Suspect Group was driven by the fact that false sensor readings have been a problem in previous space programs.

The DR uses modeled component misbehaviors to identify suspects. The drawback of this technique is that diagnoses are limited to predefined known failures in the component model. If a component fails in a mode that is not in its modeled misbehaviors, the DR cannot explain the failure properly. These unanticipated failures will be diagnosed either as multiple failures of related components or as conflicts in expected component behaviors. Conflicts between the observed component behavior and the expected modeled behavior will alert the DR to notify the user that some form of unmodeled misbehavior is taking place.

Other model-base diagnosis techniques that use nominal component behavior are not as limited in their isolation of failures because they look for conflicts between a model of the nominal system and the observed behaviors (Davis, 1988; Holtzblatt, 1989). These techniques are not currently used by the DR because they require detailed knowledge for each component, with a concomitant increase in the amount of processing power required to perform a diagnosis on a complex system. This is especially a problem when diagnosing multiple failures. It is also very difficult to build a detailed model to use these techniques for something as complex as an entire space station. The use of nominal behavior in high-level models to diagnosis failures will be an area of future research for the DR.

Using the Cold Plate example, if the Thermal Control System reports that a specific Cold Plate's outgoing temperature is high, the DR looks for possible causes to explain this behavior in a cold plate. The modeled behavior shows that a Cold Plate's high outgoing temperature could be caused by an internal failure, a low flow rate, the upstream entity having a high outgoing temperature, or a cooled component's temperature being high. As an example, if the DR sees that the observed flow rate for this specific cold plate is low, it searches the behavioral information in the Component Model for an observed behavior that causes a low flow rate. Once a root cause (suspect) is identified for the high outgoing temperature, the DR assigns this suspect to a Suspect Group within a Suspect List. The DR then checks the operational data to see if the other effects from that root cause have occurred in the predicted time frame. If they have not been observed and sufficient time has elapsed, the DR places a lower believability on that Suspect Group. Next, the DR assigns a likelihood to the Suspect Group, based on what operational reliability parameters are available for the suspect. This same process is repeated until all the components whose failure mode might cause Cold Plate 15's outgoing temperature to be high are identified and placed as Suspect Groups in the Suspect List.

### Managing the Suspect List

The Tier II managers do not observe all of the effects of a failure at the same time. Consequently, the DR will generate Suspect Lists without complete knowledge of the problem. When the DR receives the first system report, the DR responds given the available information and generates an initial Suspect List. As additional information becomes available, rather than building a completely new Suspect List, the DR first determines if the new information is related to an already-existing Suspect List. The DR relies on the expected behavioral causes and effects of identified suspects that are described in the Component Model to update, merge, or delete existing Suspect Lists generated as the result of the same failure. This yields new Suspect Lists which reflect the latest information available to the DR.

In the Cold Plate example, an initial report from the Thermal Control System may tell the DR that Cold Plate 15 has a low flow rate. The DR places Cold Plate 15 in an initial Suspect List. Later, the Communications and Tracking System might tell the DR that High Data Rate Frame Multiplexer 2 has overheated and failed. The DR notes that the High Data Rate Frame Multiplexer is cooled by Cold Plate 15, based on the Resource Supply and Demand relationships and Locality relationships in the Component Model; however, at present, there is no indication that Cold Plate 15 is overheated or will overheat. Consequently, the DR creates a second Suspect List to explain the High Data Rate Frame Multiplexer failure. Finally, the Thermal Control System reports to the DR that Cold Plate 15's outgoing temperature is high. Based on behavioral information in the Component Model, the DR finds that a low flow rate in a Cold Plate causes a high outgoing temperature in a Cold Plate, and that high outgoing temperature causes the components cooled by that particular Cold Plate to overheat. The DR relates the two failures, and merges the two Suspect Lists into one with the root cause being Cold Plate 15's low flow rate.

### Generating the Impact Sequence

The DR generates an Impact Sequence for each Suspect Group in a Suspect List. By instantiating the appropriate cause-and-effect sequences in the Impact Model, the DR dynamically constructs the cascade of effects expected to result with time from a given failure situation. The Impact Sequence is a tree-like structure made up of nodes, where the root node is an instantiation of the cause-and-effect sequence type related to the failure identified in the Suspect Group. Each node in the cascade of nodes to follow is also an instantiation of cause-and-effect sequence types that propagate from the effect identified in the particular node's parent.

The DR also attaches a severity value and an element of time to each node using severity and temporal factors in the Impact Model. The generation of the Impact Sequence is bounded by an envelope of time and severity. Given enough time, the most insignificant failure might become quite severe. Since we assume that no such failure will go unattended for a period of time sufficient to become a severe problem, the DR bounds the Impact Sequence generation by time. Similarly, generation is bounded by severity in that the DR projects the cascade of effects up to some predetermined severity value.

## Conclusions

We completed the design of the DR and the Rx, and are currently demonstrating the first, stand-alone release of the DR and the Rx. Based on experience with this prototyping effort, which includes the DR and the Rx as well as earlier efforts from which the DR and the Rx evolved (Kelly, 1988; Kelly, 1989; Marsh 1989), we learned lessons in technology transition, software reuse, and prototype software life cycles.

The primary goal of the work represented by this series of prototypes is to influence Freedom's design and implementation. Its longer-term goal is to find other domains in which this technology and approach can be used. Both are aspects of technology transition (bringing technology innovations into operational use).

Failure management is a process that is needed in many domains. The kinds of problems we are solving, and the approach we are taking to solve them, are very general. We are designing the failure managment prototype to be useable in domains beyond space operations. Examples of other possible applicable domains are NASA's manned Mars mission, chemical processing plants, or nuclear power plants.

With respect to future plans for the DR and the Rx, the two independently functioning applications will be integrated to function cooperatively. We will also integrate the DR/Rx with PI. For the commands that PI issues to be followed, the prototype must be integrated with a simulation environment for Space Station *Freedom* systems. We are investigating the possibility of developing our own simulations of the systems on a machine linked to our development workstations to provide a more portable demonstration capability.

## Acknowledgements

The work referenced in this paper was jointly sponsored by the Automation and Robotics Section under the direction of our Technical Monitor, Dennis Lawler, at NASA's Johnson Space Center (contract NAS9-18057), and by the MITRE corporation through the MITRE Sponsored Research Program. David G. Hammen and Christine M. Kelly of MITRE developed the Rx portion of the failure management prototype which is discussed in a companion article (Hammen, 1991).

## References

Baker, Carolyn, D. Hammen, C. Kelly, C. Marsh (June, 1990), "A Space Station Failure Management Prototype," presented at 1990 Space Operations, Applications and Research Symposium, Albuquerque, NM.

Davis, R., "Diagnostic reasoning based on structure and behavior," *Artificial Intelligence* 24(3), pages 347-410, North-Holland, 1984.

Davis, R. and W. Hamscher, "Model-based Reasoning: Troubleshooting," *Exploring Artificial Intelligence*, pages 297-346, Morgan Kaufmann Publishers, Inc., 1988.

Holtzblatt, L. J., R. A. Marcotte and R.L. Piazza (October 1989), "Overcoming Limitations of Model-based Diagnostic Reasoning Systems," presented at the AIAA Computers in Aerospace VII Conference.

Ireson, W., and C. Coombs, Jr., *Handbook of Reliability Engineering and Management*, New York, New York: McGraw-Hill, 1988, pp. 13.1-13.36.

Kelly, Christine M. (1988), "Automated Space Station Procedure Execution," presented at AIAA 26th Aerospace Sciences Meeting, Reno NV.

Kelly, Christine M., (1989), *Space Station Freedom Program Advanced Automation, Volume II, Series 2, Integrated Test Beds: Lessons Learned from the Data Management System Test Bed*, MITRE Technical Report Number MTR-89W00271-02, The MITRE Corporation, Houston, TX.

Marsh, Christopher and Christine Kelly (1989), Operations Management Application Prototype, Fourth Artificial Intelligence and Simulation Workshop, Detroit, Michigan, pp 75-77.

NASA (November 1989), *Space Station Projects Description and Requirements Document*, JSC 31000 Revision E, Johnson Space Center Space Station Project Office, Houston TX.

# A Failure Recovery Planning Prototype for Space Station *Freedom*

David G. Hammen and Christine M. Kelly

The MITRE Corporation
1120 NASA Road One
Houston, Texas 77058

## Abstract

NASA is investigating the use of advanced automation to enhance crew productivity for Space Station *Freedom* in numerous areas, including failure management. This paper describes a prototype that uses various advanced automation techniques to generate courses of action whose intents are to recover from a diagnosed failure, and to do so within the constraints levied by the failure and by *Freedom's* configuration and operating conditions.

*Keywords*: Planning, Failure Management, Artificial Intelligence

## Introduction

Artificial Intelligence (AI) has been successfully applied to the fields of failure diagnosis and plan generation. This paper addresses the application of AI planning techniques to failure recovery. The Recovery expert (Rx) is an existing prototype that generates plans whose intents are to recover from diagnosed failures on the Space Station *Freedom* and to do so within the constraints levied by the failure and by other current operating conditions. The Rx uses artificial intelligence techniques (such as pattern matching, symbolic reasoning, and goal-directed planning) and statistical techniques (multiple criteria decision making) to generate the courses of action.

### Background

The National Aeronautics and Space Administration (NASA) is in the process of designing and implementing a permanently manned space station, Space Station *Freedom*, to be put into low earth orbit. *Freedom* has a hierarchical, distributed command and control architecture. The highest level in the architecture is concerned with global, vehicle-wide issues and includes the people controlling *Freedom* and the software that they use to do their jobs. Our failure management prototype, initially described in (Hammen, Baker, Kelly, and Marsh, 1990) and expanded upon in this report, demonstrates the applicability of expert system software to provide decision support to *Freedom's* crew in recovering from vehicle failures.

This prototype is implemented in ART/Ada™, TAE™ Plus, and Ada, and comprises the Diagnostic Reasoner (DR), which diagnoses and analyzes the failure, and the Rx, which formulates courses of actions to take for recovery (Baker, Hammen, Kelly, and Marsh, 1991). This report focuses on the Rx; a companion article (Baker and Marsh, 1991) focuses on the DR. The DR and Rx are not completely automated functions: they are decision aids, and the user is provided with interaction capabilities throughout the failure management process.

The key technologies used in the implementation are model-based reasoning (for the DR) and goal-directed planning (for the Rx). The objective of the prototype is to show how we use these technologies and how the technologies work together synergistically. By demonstrating the value of this approach to failure management we hope to influence how *Freedom* failure management software will eventually be implemented.

Figure 1 shows an overview of the environment in which the DR and Rx prototypes work. The DR receives status and configuration reports that reflect changes in the operational components. In the case of a failure, the DR diagnoses the failure and the Rx formulates a set of possible solutions to the failure. Once a specific Course of Action has been selected, another prototype not described here, the Procedures Interpreter, sends commands that implement the selected Course of Action.

## Failure Management Terminology

A fault is a "known or hypothesized defect in hardware or software causing error" (MDSSC, 1989). A fault may manifest itself as a failure, which is "the inability of a system, subsystem, component, or part to perform its required function" (MDSSC, 1989).

## DR Overview (Baker and Marsh, 1991)

When the DR receives reports that indicate a failure it looks for possible causes, or suspects. More than one possible cause might be proposed for a given set of symptoms, and more than one independent fault might be proposed as a possible cause. The DR projects the effects of the possible causes into the future, producing failure modes and effects criticality analyses, or Impact Sequences. These provide insight into the severity of future impacts and allow the Rx to recognize and mitigate the more severe problems first. The possible causes and their impacts are identified in a Suspect List, which is sent to the Rx.

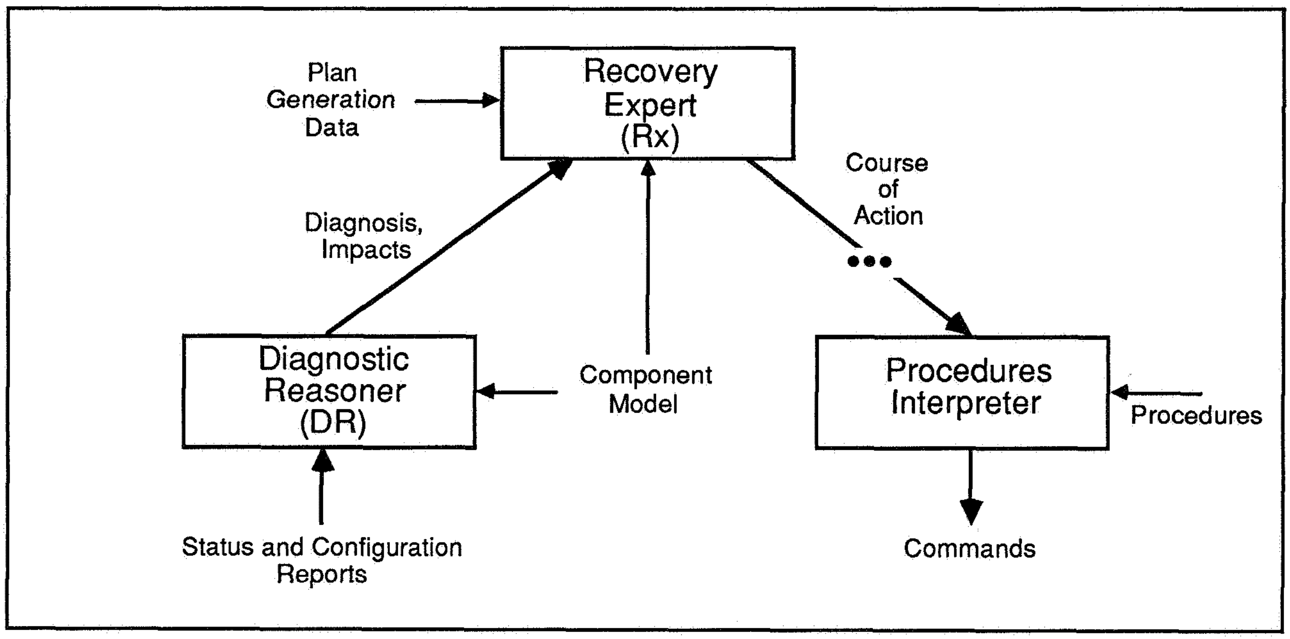


**Figure 1. Overview of DR/Rx Process Flow**

## Rx Overview

The Rx bases its approach to the problem on supplied diagnoses and corresponding failure modes and effects criticality analyses. The Rx selects ways in which to attack the problem and develops partial plans that address specific aspects of the problem based on these chosen attacks. These partial plans are merged to generate comprehensive plans that address the entire problem. The Rx generates multiple plans, good and not so good, for further evaluation by the user. An overview of the processing performed by the Rx is presented in figure 2.

The Rx selects attacks on the problem based on the soundness of the diagnoses. When a diagnosis is sound the Rx attempts to develop primary plans that directly attack the root of the problem. When the diagnosis is equivocal the Rx attempts to generate primary plans whose intents are to collect information that could refine future diagnoses. For preliminary diagnoses the best response might be to do nothing. The Rx may also generate secondary plans that mitigate potential downstream impacts of the failure, allowing adequate time to realize the primary plans.

For each attack that is selected the Rx attempts to build partial Courses of Action. Initially, a top-level goal is generated based on the problem and the selected attack. This goal triggers the goal-directed planning process, which generates a tree-like structure of goals and activities. An activity specifies how to use a pre-defined procedure to achieve the goal of the activity. The procedures have prerequisites that lead to sub-goals, extending the tree. Partial Courses of Action are

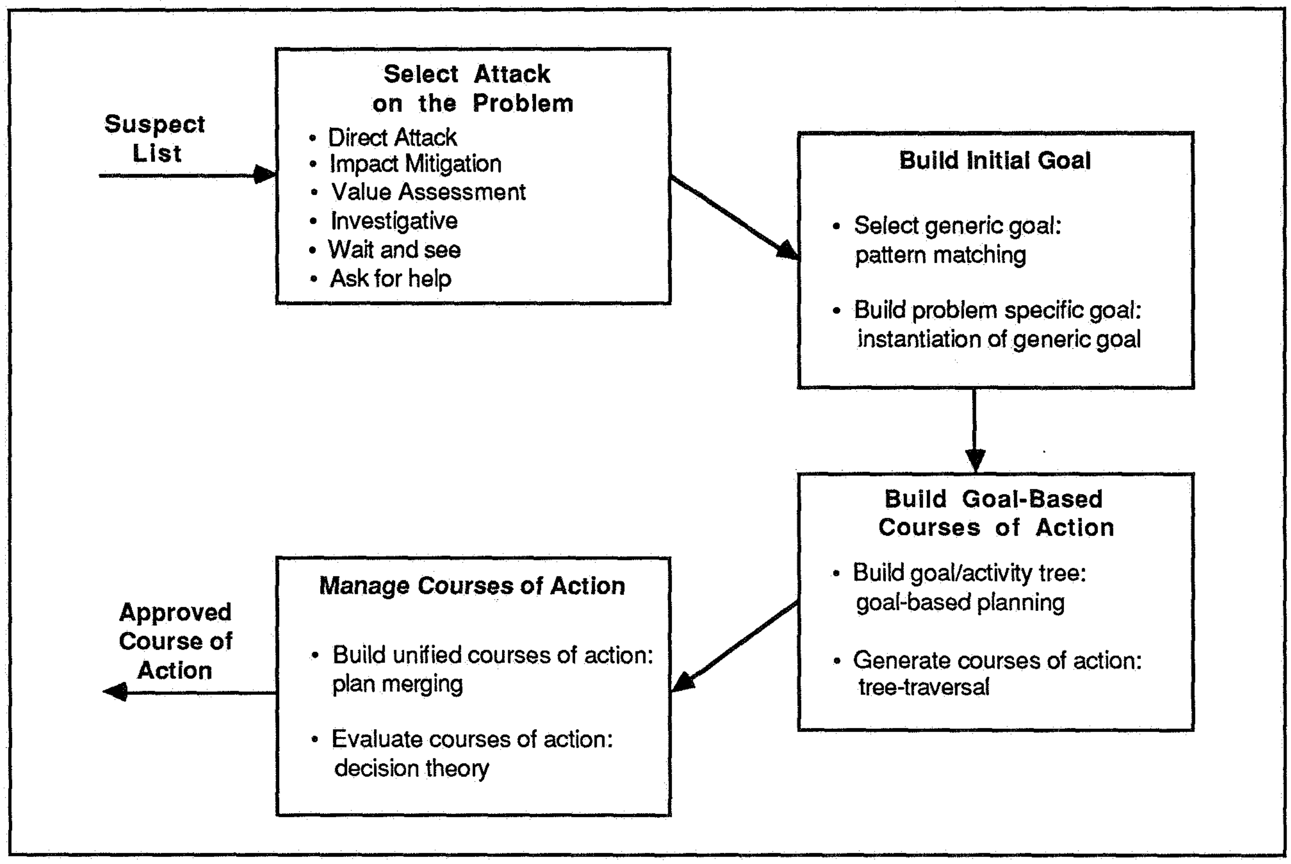

Suspect List
Select Attack on the Problem
• Direct Attack
• Impact Mitigation
• Value Assessment
• Investigative
• Wait and see
• Ask for help
Build Initial Goal
• Select generic goal: pattern matching
• Build problem specific goal: instantiation of generic goal
Build Goal-Based Courses of Action
• Build goal/activity tree: goal-based planning
• Generate courses of action: tree-traversal
Manage Courses of Action
• Build unified courses of action: plan merging
• Evaluate courses of action: decision theory
Approved Course of Action

**Figure 2. Overview of Rx Process Flow**

extracted from this tree using tree traversal techniques. Follow-up actions that eliminate undesirable side effects of the activities can be added, again using goal-directed planning techniques.

Each partial Course of Action addresses only a part of the problem. The final action that must be taken before a recovery plan can be designated as complete is to merge partial Courses of Action, creating comprehensive Courses of Action that address the root of the problem, mitigate the side effects of the problem, and do so within the constraints levied by the problem.

### Paper Overview

The above summary of what the Rx does provides little indication of how it does it. We intend in this paper to describe how the Rx interacts with external agents and how it produces Courses of Action in response to diagnosed faults. Since the DR and Rx are intended to reason about a dynamic system, we begin with a description of the mechanism used for representing and reasoning about relative time. In the following sections we describe the interactions of the Rx and external agents in terms of the data that are exchanged between the Rx and those agents, and we describe the means by which the Rx forms Courses of Action in terms of the processes used to build and evaluate the Courses of Action.

## Relative Time Representation

The DR and Rx reason about a dynamic system where the state of the system changes with time. Timing and temporal relationships between detected events, projected future events, and planned activities are important to the functioning of the software. The problem here is not how to express the specific, absolute time at which detected events have occurred but how to express the relative timing between events: this is a matter of timing rather than time. We express relative time on a logarithmic scale of seconds. We chunk the time into discrete intervals by truncating to an integral value because this enables reasoning about time qualitatively and because this represents the limits of our accuracy. We terminate the scale at 0 since very small time intervals are not within the scope of the DR or Rx. This time scale is depicted in figure 3. An example of the use this scale is the time interval between an event and an expected impact of the event. If the time interval is 0 the impact is expected to be felt within ten seconds, and if it is 2 the impact should be felt in 100 to 1000 seconds. In discussions of our knowledge representation any reference to timing is made to this scale.

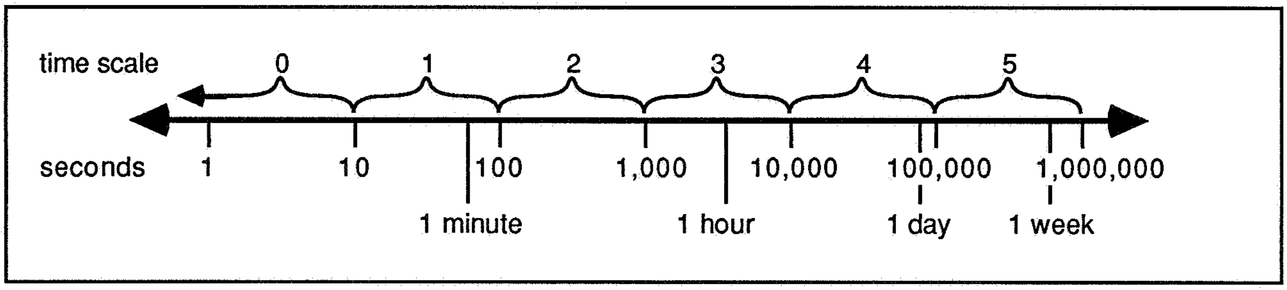


**Figure 3. Relative Time Scale**

## Externally Visible Data

In this section we describe those types of data that the Rx exchanges with external agents: the Component Model, which describes the status and structure of *Freedom's* components; the Suspect Lists, which contain the diagnoses generated by the DR and cause the Rx to take action; the Procedure Metadata, which describe the actions that can be taken; and the Courses of Action that describe the actions to be taken.

### The Component Model

The Component Model, which is at the heart of processing for the DR, also plays a significant role for the Rx. The Component Model incorporates dynamic structural and status information as well as other information that is not used by the Rx. The structural information describes the physical and functional relationships between components. Figure 4 illustrates two of the several types of inter-component relationships expressed in the Component Model. The status information describes the components' modes of operation, their health, and key operational values related to their behavior. Figure 5 illustrates some of the dynamic status information that is collected for a component instance. The Rx uses the structural and status information to tailor the Courses of Action it generates to *Freedom's* configuration and operating conditions. For example, the Rx would reject a Course of Action that involves the use of a specific component that had previously failed.

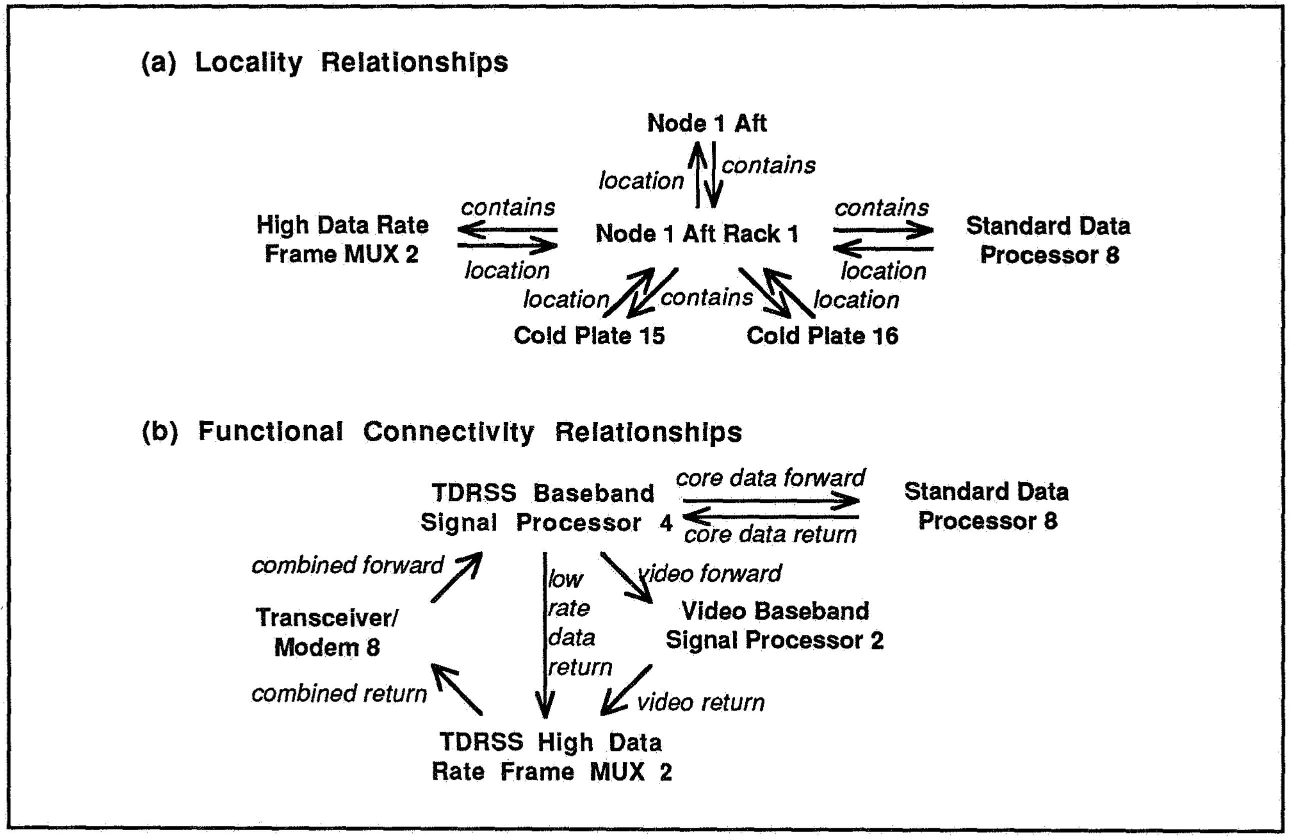


**Figure 4. Component Model Structural Information**

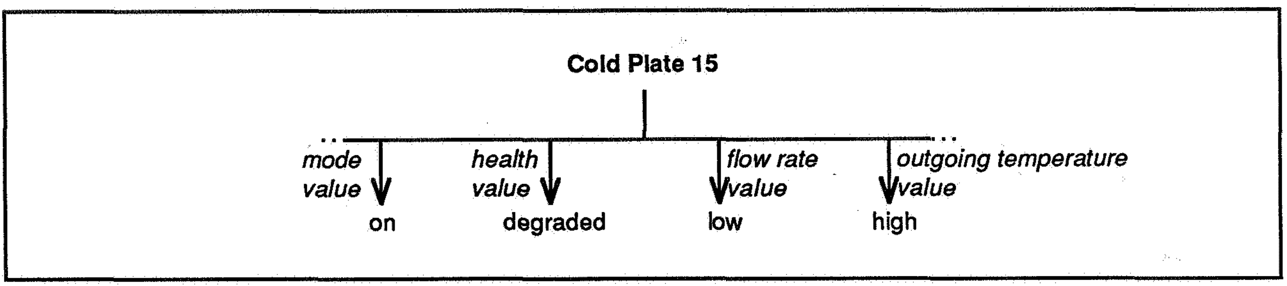


**Figure 5. Component Model Status Information**

The Component Model describes the components, the functions they provide, and where they are located. The components, functions, and locations are all described hierarchically, beginning with abstract object types that expand into more and more precise object types, and terminating in specific instances of object types. Each object class typically has many individual instantiations. This representation also encompasses information about characteristics of the components; we used inheritance techniques to reduce the amount of explicitly specified information.

## Suspect List

A Suspect List identifies a set of related observed (off-nominal) behavior attribute values and those components whose off-nominal performance could result in those observations. Figure 6 shows the structure of a Suspect List.

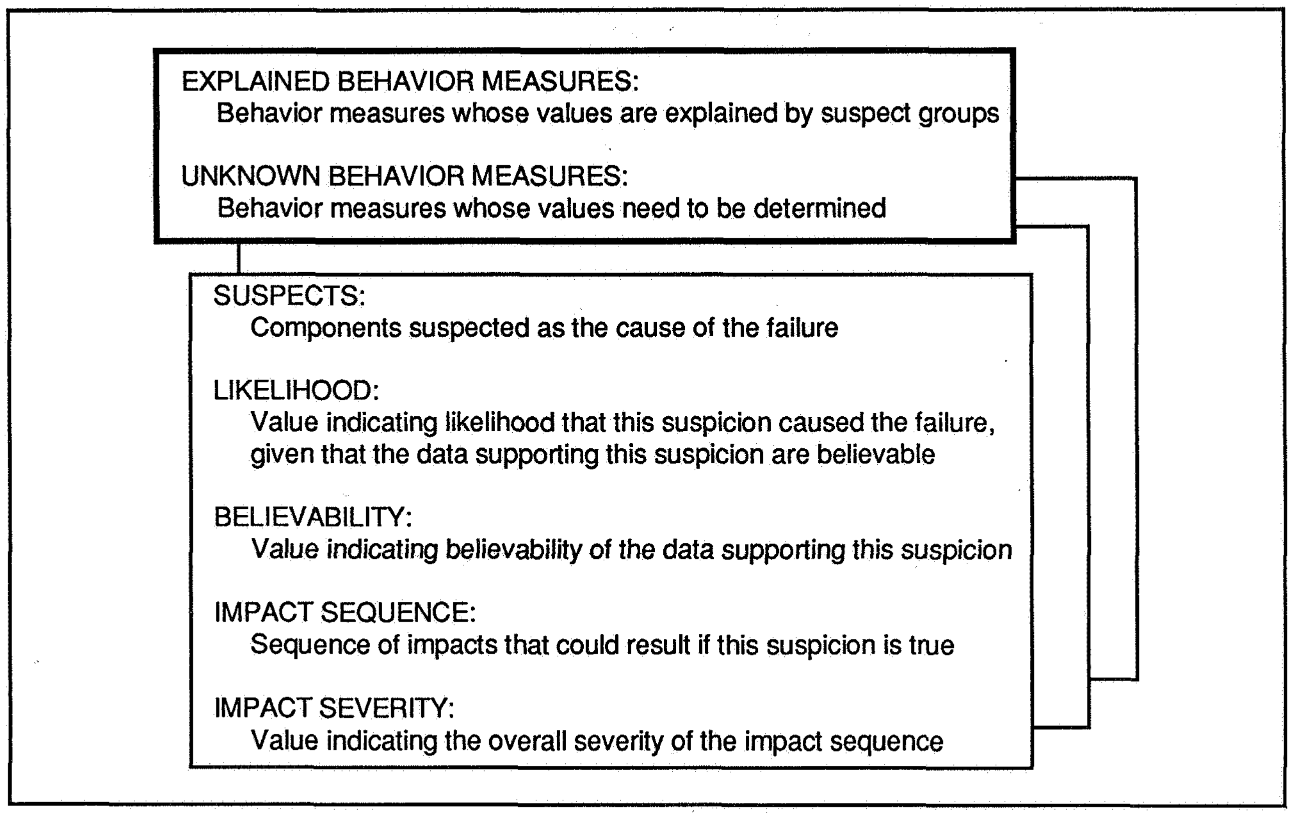
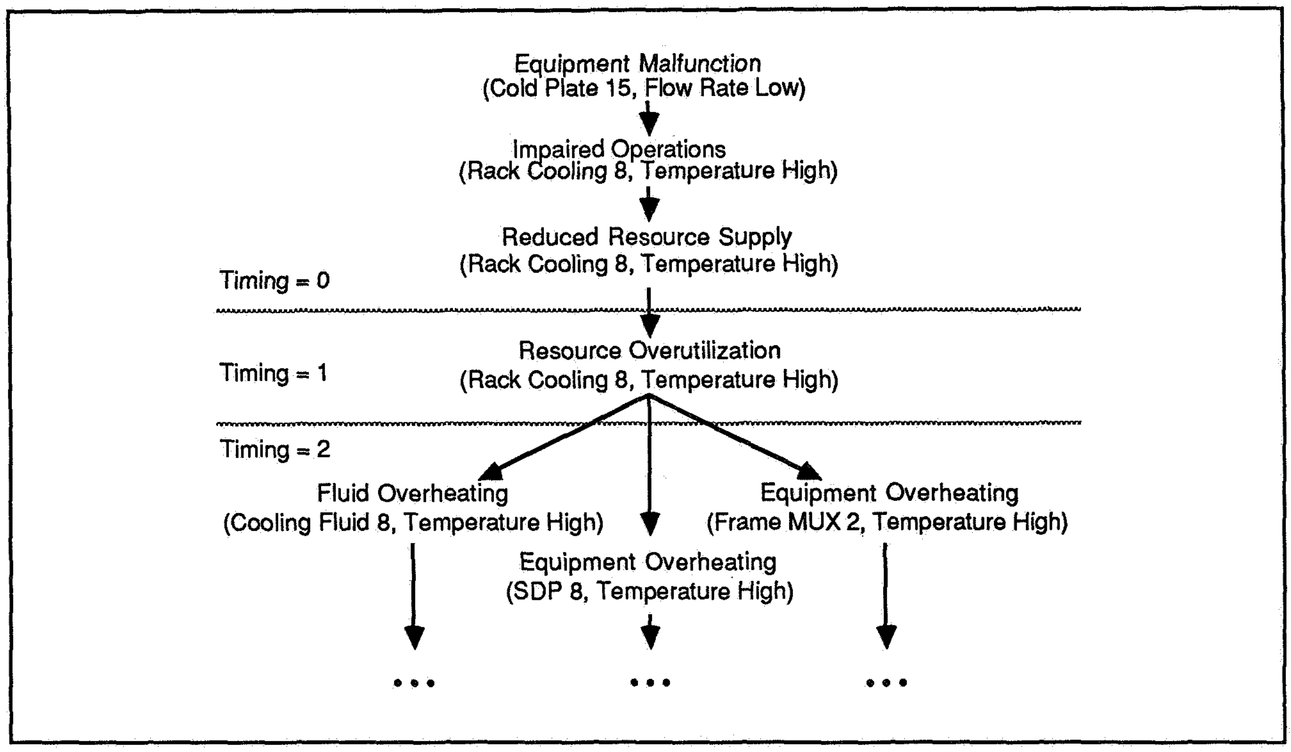

EXPLAINED BEHAVIOR MEASURES:
Behavior measures whose values are explained by suspect groups
UNKNOWN BEHAVIOR MEASURES:
Behavior measures whose values need to be determined
SUSPECTS:
Components suspected as the cause of the failure
LIKELIHOOD:
Value indicating likelihood that this suspicion caused the failure, given that the data supporting this suspicion are believable
BELIEVABILITY:
Value indicating believability of the data supporting this suspicion
IMPACT SEQUENCE:
Sequence of impacts that could result if this suspicion is true
IMPACT SEVERITY:
Value indicating the overall severity of the impact sequence

**Figure 6. Suspect List Structure**

A Suspect List is associated with a specific failure. The DR might find that more than one possible cause explains the same failure, in which case each explanation is reported as a separate diagnosis, or Suspect Group, in the same Suspect List. The DR might find that several independent component faults are required to explain the failure, in which case all the independent faults will be identified as a possible cause in a single Suspect Group.

A Suspect List also can identify key unknown behavior measures: operational behavior measures whose values are not available directly. The assessment of some behavior measures will require special resources (such as asking an astronaut to execute a procedure to collect data) or will induce inter-system interactions (such as taking a component offline to perform diagnostics). When the DR encounters one of these unknown measures along one of its diagnostic causal pathways it will post the measure in the Suspect List as an unknown measure whose assessment should help refine the diagnosis.

The DR adds likelihood and believability values and an Impact Sequence and its severity to each Suspect Group. Likelihood is an estimate of the probability that the Suspect Group caused the failure: for example, a component that fails frequently is more likely to be the cause than one that fails rarely. Believability indicates how reliable the data are that are used to determine the Suspect Group as the cause of failure: for example, some sensors are known to fail frequently or give unreliable readings. An Impact Sequence expresses the cascade of effects that might result from a given diagnosis. Each impact in the sequence is tagged with timing and severity information, and is an instance of a generic impact type. This allows the Rx to reason about the Impact Sequence generically and specifically, and allows the Rx to address the most acute or most severe impacts first. Figure 7 illustrates a portion of an Impact Sequence.



Equipment Malfunction
(Cold Plate 15, Flow Rate Low)
Impaired Operations
(Rack Cooling 8, Temperature High)
Reduced Resource Supply
(Rack Cooling 8, Temperature High)
Timing = 0
Resource Overutilization
(Rack Cooling 8, Temperature High)
Timing = 1
Timing = 2
Fluid Overheating
(Cooling Fluid 8, Temperature High)
Equipment Overheating
(Frame MUX 2, Temperature High)
Equipment Overheating
(SDP 8, Temperature High)
• • •
• • •
• • •

**Figure 7. Impact Sequence Segment**

## Procedure Metadata

The Rx is able to build a Course of Action because extensive information about the pre-defined procedures is incorporated into its knowledge base. The Procedure Metadata describe how, why and under what constraints and conditions a procedure is used. These data are used to generate and evaluate a Course of Action. Examples of the Procedure Metadata are presented in table 1.

Table 1. Procedure Metadata

| Procedure Name | Entry Conditions | Pre-requisites | Exit Conditions | Post-requisites | Motives | Schedul-ability | Timing |
|---|---|---|---|---|---|---|---|
| Test Cold Plate | Cold Plate Mode On | None | None | None | Cold Plate Health Suspect | 3 | 2 |
| Replace Cold Plate | Cold Plate Health not Nominal | Cold Plate Fluid level Empty | Cold Plate Health Nominal | Test Cold Plate | None | 3 | 3 |
| Drain Cold Plate | None | Cold Plate Mode Off | Cold Plate Fluid level Empty | None | None | 3 | 2 |

Much of the Procedure Metadata is related to the goals associated with the procedure. A goal describes some aspect of the world state, such as the status of the health of a component. The entry conditions define what conditions must be true before a procedure is performed. The prerequisites define what conditions need to be made true before a procedure can be started. The exit conditions define what should be true when the procedure is complete (in other words, the goal of the procedure). The post-requisites define what needs to be done after completing the procedure. The motives of the procedures define what conditions would be true to lead one to believe that this procedure is one that applies. Schedulability is an a priori estimate of how long it will take before the procedure can start executing, while timing is an estimate of how long it will take to execute the procedure once it has started.

## Course of Action

A Course of Action identifies the activities to be performed, temporal relationships between the activities, and entry conditions that should be true prior to performing the activities. These entry conditions provide a check against run-away operations: the execution of the Course of Action will be stopped if the entry conditions to an action are not met. Table 2 portrays a Course of Action for an equipment malfunction of a cold plate.

Table 2. A Sample Course Of Action

| Procedure Name | Parameters | Temporal Relationships | Entry Conditions |
|---|---|---|---|
| Switch to Backup | Cold Plate 15 | None | Cold Plate 16 Mode On |
| Valve Off Cold Plate | Cold Plate 15 | Start after end of Switch to Backup | Cold Plate 16 Status In use |
| Drain Cold Plate | Cold Plate 15 | Start after end of Valve Off Cold Plate | Cold Plate 15 Mode Off |
| Replace Cold Plate | Cold Plate 15 | Start after end of Drain Cold Plate | Cold Plate 15 Fluid level Empty |
| Test Cold Plate | Cold Plate 15 | Start after end of Valve On Cold Plate | Cold Plate 15 Mode On |

## Rx Processing

The Rx is responsible for building and recommending Courses of Action that, when executed, should result in recovery from the problem diagnosed by the DR. The Rx uses the pre-defined crew procedures as the elements from which it builds a Course of Action. A Course of Action includes actions that realize the recovery but also could include intermediate actions that mitigate the more acute consequences of the problem, providing adequate time to realize the recovery itself. A Course of Action should address the entire problem and should do this within the constraints levied by the problem. A future goal is to feed user-selected Courses of Action to the existing Procedures Interpreter prototype, which would drive simulations of various *Freedom* systems. This will provide a closed loop operation between the failure management and the systems prototypes, making the entire prototype operation more realistic.

The Rx performs four basic operations in its generation of a Course of Action, as depicted in figure 2: it selects attacks on the problem, it generates goals that address the problem in concert with the chosen attack, it builds Courses of Action that achieve these goals, and it manages and merges Courses of Action, resulting in user-approved comprehensive Courses of Action that address the entire problem. In the remainder of this section, we discuss the operations performed by the Rx to build the Courses of Action.

### Selecting an Attack

Several options are available for dispositioning the diagnosis reported by the DR. The Rx must determine how to attack the problem based on the conditions in place at the time the failure occurs. The types of attacks that can be devised are direct attack, information gathering (value assessment and investigative), impact mitigation, wait and see, and ask for assistance.

#### Types of Attacks

The DR unequivocally and unerringly identifies some failures; in these cases, the problem can be addressed directly. For example, a known failed cold plate could be replaced by a crew member. In fact, a direct attack is the only approach that develops a Course of Action that solves the root cause of the problem. Even if other steps are taken to defer critical conditions from developing, eventually a direct attack on the problem must be taken to prevent manifestations of the problem from recurring again in the future.

Additional information could help the DR to refine diagnoses that are in doubt. The DR may indicate (in the Suspect List) that it needs to know the value of a behavior measure whose value is currently unknown; in these cases, the Rx will attempt to build a value assessment Course of Action whose intent is to determine the behavior measure's value. For example, a cold plate's flow rate may not be directly available but needs to be assessed, or a crew member may be asked to observe whether a specific component is exhibiting behavior that is observable by a person but not measurable by a sensor, such as whether a component is vibrating. Another approach must be taken when there are no unknown values that could obscure the cause of the problem; in these cases, there may be an existing diagnostic procedure that could help refine the diagnosis, and the Rx will use such a procedure to form the basis of an investigative Course of Action. For example, a procedure that reconfigures the way that communications components are connected could be used as a basis for determining whether a given component is functioning properly.

The Rx is also concerned with assuring that the actions taken are sensible in light of the foreseeable impacts. When severe or acute downstream impacts could occur, the Rx will develop plans that mitigate these downstream impacts so that the desired action can be sensibly performed. For example, it may take a long time to replace a failed cold plate but moderate impacts such as damage to equipment cooled by the cold plate could occur in a short time and severe impacts such as the loss of a node sometime later. The Rx will develop plans (for example, equipment reconfiguration) to ensure that these impacts do not occur during the time needed to realize the recovery.

It might be imprudent for the Rx to begin generating plans when a preliminary Suspect List is reported by the DR: the costs of probably doing the wrong thing and of wasting valuable computer time to derive this unneeded Course of Action outweigh the benefits of possibly doing the right thing. In these cases, the Rx will do nothing, taking a wait and see attitude until the DR has observed some predicted near-term downstream impacts and revised its diagnoses.

Finally, the Rx will request operator intervention if it cannot find an adequate response. Since the DR and Rx are being implemented as decision support tools, the capability for the operators to interact with the processing is available at all decision-making points in the processing.

### Approach

Selecting an attack involves evaluating the diagnosis according to a number of criteria and then weighing the alternatives based on these potentially conflicting evaluations. The criteria include

- the unequivocalness of the diagnosis;
- the immediacy and severity of the downstream impacts;
- the need for assessing an unknown behavior measure value;
- the cost of developing a Course of Action; and
- the surety of the diagnosis, based on
    - a statistical estimate of the likelihood of failure in the diagnosed failure mode(s) and
    - the believability of the reports that led to the diagnosis.

Techniques from decision theory (Markland, 1983; North, 1977) are designed to handle the quantitative nature of the evaluations that need to be made and some of these techniques explicitly handle making decisions under uncertainty. We use decision analysis techniques to determine the expected utility of an attack. Those attacks whose utility exceeds some preset value are released for Course of Action generation.

This thresholding allows an indeterminate number of attacks to be made. The intent of selecting an attack is not to select the single very best attack but to select the most appropriate attacks. Each of the possible attacks can yield partial Courses of Action that address a specific aspect of the problem. To form a comprehensive Course of Action that addresses the entire problem some of these partial Courses of Action would then have to be merged (see the section "Managing Courses of Action"). Thus there often is not a best attack; the best approach instead is to make several separate attacks on the problem in a coordinated manner. Another aspect of managing Courses of Action is to select the best Course of Action; such a selection is best made when a reasonable number of options are available, and when the process of selecting attacks works well a reasonable number of options are generated for further evaluation.

## Generating Goals

The Rx develops goals that address the failure in concert with the chosen attack. These initial goals trigger the goal-directed planner to build a partial Course of Action that achieves the goal. Several goals may have to be created and expressed as partial Courses of Action that are then merged to create a comprehensive Course of Action that addresses the entire problem.

Generic workarounds to impacts are used to generate goals for direct attacks and impact mitigation attacks. (The fault is always represented as the initial impact in the Impact Sequence, so a direct attack can be viewed as a special case of impact mitigation.) Selected impacts in the DR-generated Impact Sequence are matched with generic workarounds to generate goals concerning the exit conditions of procedures. For example, the Rx could address a reduction in cooling capacity to a rack as a special case of a resource supply reduction. A generic workaround for a resource supply reduction is to decrease resource utilization. Applying this generic goal to the specific problem results in the specific goal of reducing the cooling load at the affected rack. After the goals are generated, the Course of Action builder will look for procedures whose exit conditions meet the goals. Sample goal generation data are presented in table 3.

**Table 3. Goal Generation Information for Impact Mitigation**

| Impact | Workarounds | Goal Generation Information |
|---|---|---|
| Equipment Malfunction | Repair or Replace Equipment | Equipment Health Nominal |
| Impaired Operations | Use Backup Capability | Backup Capability Status In-use |
| | Use Alternate Capability | Alternate Capability Status In-use |
| Reduced Resource Supply | Augment Resource Supply | Resource Level Nominal |
| Resource Overutilization | Augment Resource Supply | Resource Level Nominal |
| | Decrease Resource Utilization | Resource Consumption <= Resource Level |

Each impact type can have several alternative workaround types. It is not necessarily the case, however, that each workaround type will be available for a given instance of an impact. For example, if there is a reduced resource supply of cooling, a workaround to that impact type is "Use Alternate Capability". In the general case of cooling, an alternate capability might be to turn on a fan when the air conditioner is not working to full capacity. In the case of a cold plate, however, there is no such alternate capability: a cold plate is the only source of cooling available. The Rx must determine the applicability of the workarounds to the situation being examined.

Goals are expressed symbolically (for example, Backup Equipment Status In-Use) in the goal generation data. Once a workaround is determined, the Rx creates a specific instance of a goal (such as Cold Plate 16 Status In-Use, where Cold Plate 16 is the backup unit to the cold plate exhibiting off-nominal behavior).

While a procedure's exit conditions are useful for finding a procedure with an applicable outcome, a procedure's motives are useful for finding a procedure that can be used when a given set of observations are true. The Rx builds goals for data collection and diagnostic attacks that cause the goal-directed planner to select procedures whose motives are to collect the required data items, to narrow down the number of suspect groups, or to verify an uncertain diagnosis.

## Building Courses of Action

The Rx uses goal-directed planning (Chapman, 1987; Wilkins, 1989) to build a goal-activity directed graph. This graph does not explicitly specify the Courses of Action; these are extracted from the graph using graph traversal techniques.

Goal-directed planning matches goals with goal operators to build a plan. For the Rx, the goal operators are the pre-defined procedures. Unless a goal is already satisfied (the Rx checks for this in the Component Model) the Rx searches through the Procedure Metadata to find the procedure(s) that might accomplish the goal. Procedures can have prerequisites; when they do, this generates additional goals that need to be met, and the Rx once again searches the Procedure Metadata to find procedures that might accomplish these goals. A procedure can have more than one prerequisite, and these can be conjunctive, possibly requiring several procedures to accomplish them, or disjunctive, possibly creating alternate approaches to accomplishing them. The goal generation–goal accomplishment process iterates, resulting in the goal-activity graph, a tree-like structure of goals and activities, with the activities specifying how the operators are to be invoked. The graph building process obeys the constraints imposed by the failure and its impacts as well as constraints imposed by other operational conditions (such as configuration). Figure 8 shows a portion of a goal-activity graph built in response to an equipment failure of Cold Plate 15.

Even in our simplistic test application we found that an activity can simultaneously accomplish several goals. The basic goal-directed planning techniques outlined above would generate a separate activity for each goal. It would be foolish, and possibly erroneous, to perform the underlying procedure once for each goal. Instead, we devised techniques to identify such homologue activities—these are activities that have the same underlying procedure with the same parameters, which, when executed, will accomplish the goals of all of the homologue activities. Similarly, goals can have homologues, and we also devised techniques to identify such homologue goals; for example, multiple procedures might have the same prerequisite. Homologue activities or goals are not just identical but represent one single activity or goal (in comparison, identical twins are identical but are two separate individuals), so it is only necessary to process one of the homologues. A side effect of this is that the goal-activity graph is a graph rather than a tree.

The goal-activity graph does not directly specify a Course of Action. Graph traversal techniques must be used to extract the viable Courses of Action from the graph. The graph terminates in activities with no prerequisites, activities with unsatisfiable prerequisites, and goals that are already satisfied, and the Courses of Action for these are trivial. We extract the Courses of Action that achieve the top-level goal by building from the trivial Courses of Action at the bottom of the graph and working up to the top-level goal. The Courses of Action specify the activities to be performed and the temporal relationships between them. The procedures identified by the activities, when executed, should achieve the specific goal that the Course of Action addresses.

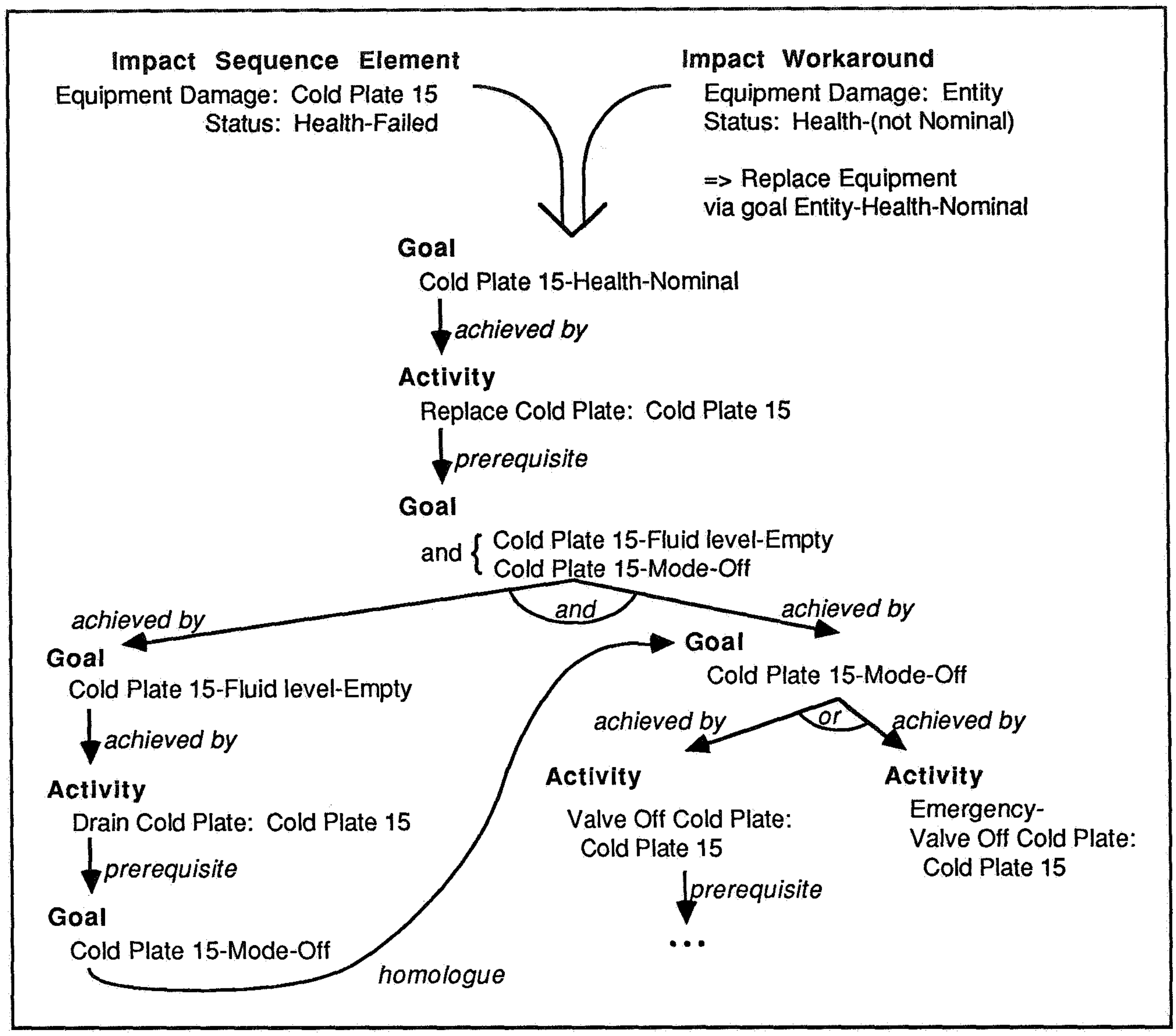


**Figure 8. Portion of a Goal-Activity Graph**

Developing a plan in a reasonable amount of time requires the judicious use of search and reasoning capabilities. The Rx uses procedure schedulability, timing, likelihood of success, and undesirability estimates to label some search paths as fruitless. The user can redirect the Rx to investigate these terminated paths.

## Managing Courses of Action

The Rx can build multiple Courses of Action in response to a single problem. For example, the desired action should be achieved by a replacement Course of Action, but several impact mitigation Courses of Action will be required for this desired action to have a successful outcome. These multiple Courses of Action must be merged and ordered to form a comprehensive Course of Action that addresses the entire problem rather than a portion of the problem. The attempt is to build Courses of Action that solve the root problem within the constraints levied by the failure.

The Rx attempts to develop several alternate means of addressing the problem. These alternate Courses of Action must be evaluated prior to execution. This final evaluation is performed using techniques from decision theory by a separately contracted prototype, the Technique for Rapid Impact Analysis and Goals Evaluation (TRIAGE) (Krupp and Burke, 1991). We have coordinated efforts with the TRIAGE implementors to allow for the integration of these functions.

## Conclusions

The design of the Rx has been completed and an initial demonstration is available. Based on experience with this prototyping effort, including the Rx as well as earlier efforts from which it evolved, we have learned lessons in technology transition, software reuse, and prototype software life cycles. These lessons are discussed in (Baker, Hammen, Kelly, and Marsh, 1991).

This prototyping effort is not yet complete. The first step that we took in implementing an integrated prototype was to develop the DR and Rx as separate standalone applications. During this phase, we took care to express shared knowledge using the same structure. This phase has been completed, and initial demonstrations of the standalone DR and Rx are available. Our next step is to integrate the DR and Rx (over a network if necessary) to function cooperatively. Interfaces between the applications have already been defined; this step of integration merely implements these interfaces.

An important area of technical concern when implementing expert systems in a constantly changing environment has to do with coordinating past and current conclusions drawn by the software with the ongoing changes. Failures and their effects occur over a span of time. Consequently, the DR might identify suspects without complete knowledge of the problem, but it can update these diagnoses as additional information becomes available. The Rx might generate preliminary Courses of Action that can be overridden by Courses of Action generated in response to more accurate diagnoses. This association between current and past reports confronts most perpetually executing real-time monitoring systems, a class of programs to which the DR and Rx belong. The Rx does not yet take advantage of the correlations between the current and previously generated Suspect Lists; we plan to make the Rx utilize this information. We also plan to implement reuse of previously derived Courses of Action by extending the existing homologue detection capability.

The concepts underlying the DR and Rx have applicability beyond Space Station *Freedom,* the target environment for the initial prototypes. With this extensibility in mind, we separated the descriptions of the objects (such as procedures and components) from the generic capabilities of representing and reasoning about the objects. By formalizing this separation and by creating tools to develop and manage the application-specific objects we can create a failure management shell that could be applied to many domains that are hierarchical and are operated through pre-defined procedures.

## Acknowledgements

The work referenced in this paper was jointly sponsored by the Automation and Robotics Section at NASA's Johnson Space Center under contract NAS9-18057 and by the MITRE Corporation through the MITRE Sponsored Research Program.

We thank Chris Marsh and Jayne Baker, our coworkers and the authors of the companion to this article describing the DR (Baker and Marsh, 1991), and Dennis Lawler, our NASA project monitor, who participated in the design of the Rx and critiqued its development.

## References

Carolyn G. Baker, David G. Hammen, Christine M. Kelly, Christopher A. Marsh, 1991, *The Operations Management Application Failure Management Prototype*, WP-91W00006, The MITRE Corporation, Washington, D.C. (in press).

Carolyn G. Baker and Christopher A. Marsh, May 1991, "A Failure Diagnosis and Impact Assessment Prototype for Space Station *Freedom*," Paper presented at the 1991 Goddard Conference on Space Applications of Artificial Intelligence, Greenbelt MD.

David Chapman, July 1987, "Planning for Conjunctive Goals," *Artificial Intelligence*, Vol. 32 pp 333-377, reprinted in *Readings in Planning*, James Allen et al. eds., San Mateo, California: Morgan Kaufmann Publishers, Inc., pp 537-558.

David Hammen, Carolyn Baker, Christine Kelly, and Christopher Marsh, June 1990, "A Space Station Failure Management Prototype: DR and Rx," Paper presented at the 1990 Space Operations, Applications and Research Symposium, Albuquerque, NM.

Joseph C. Krupp and Thomas E. Burke, January 1991, *TRIAGE: Technique for Rapid Impact Analysis and Goals Evaluation*, DSA Report Number 84/1197, Decision-Science Applications, Inc., Arlington, VA.

Robert E. Markland, 1983, "Decision Analysis," *Topics in Management Science*, Second Edition, New York, New York: John Wiley & Sons, Inc., pp 790-824.

McDonnell Douglas Space Systems Company (MDSSC), September 1989, *Failure Tolerance and Redundancy Management Design Guide for Space Station Work Package 2 Systems and Elements*, MDSSC, Huntington Beach, CA.

D. Warner North, September 1968, "A Tutorial Introduction to Decision Theory," *IEEE Transactions on Systems Science and Cybernetics*, SSC-4:3, reprinted in *Readings in Uncertainty Reasoning*, San Mateo, CA: Morgan Kaufmann Publishers, Inc., pp 68-78.

David E.Wilkins, 1988, *Practical Planning: Extending the Classical AI Planning Paradigm*, San Mateo, CA: Morgan Kaufmann Publishers, Inc.

N91-22779

# The Generic Spacecraft Analyst Assistant (GenSAA): A Tool for Automating Spacecraft Monitoring with Expert Systems

**Peter M. Hughes**
*NASA/Goddard Space Flight Center*

**Edward C. Luczak**
*Computer Sciences Corporation*

## Abstract

Flight Operations Analysts (FOAs) in the Payload Operations Control Center (POCC) are responsible for monitoring a satellite's health and safety. These analysts closely monitor real time data looking for combinations of telemetry parameter values, trends, and other indications that may signify a problem or failure. As satellites become more complex and data rates increase, FOAs are quickly approaching a level of information saturation.

The FOAs in the spacecraft control center for the COBE (Cosmic Background Explorer) satellite are currently using a fault-isolation expert system named the Communications Link Expert Assistance Resource (CLEAR), to assist in isolating and correcting communications link faults. Due to the success of CLEAR and several other systems in the control center domain, many other monitoring and fault-isolation expert systems will likely be developed to support control center operations during the early 1990s.

To facilitate the development of these systems, a project has been initiated to develop a domain-specific tool, named the Generic Spacecraft Analyst Assistant (GenSAA). GenSAA will enable spacecraft analysts to easily build simple real-time expert systems that perform spacecraft monitoring and fault isolation functions. Expert systems built using GenSAA will assist spacecraft analysts during real-time operations in spacecraft control centers.

This paper describes lessons learned during the development of several expert systems at Goddard, thereby establishing the foundation of GenSAA's objectives and offering insights on how problems may be avoided in future projects. This will be followed by a description of the capabilities, architecture, and usage of GenSAA along with a discussion of its application to future NASA missions.

## Introduction

NASA's Earth-orbiting scientific satellites are becoming increasingly sophisticated. They are operated by highly trained personnel in the mission's Payload Operations Control Center (POCC). Currently at the Goddard Space Flight Center (GSFC), missions utilize either a dedicated control center (e.g. LANDSAT and the Hubble Space Telescope) or share resources in the Multi-Satellite Operations Control Center (e.g. Cosmic Background Explorer and the Gamma Ray Observatory). In either case, POCC personnel called Flight Operations Analysts (FOAs), are responsible for the proper command, control, health, and safety of the satellite.

The satellite control centers operate round-the-clock throughout the lifetime of the spacecraft. There are typically multiple real-time communications events daily with each satellite. During these events, the FOAs must:

– establish and maintain the telecommunications link with the spacecraft,
– monitor the spacecraft's health and safety,
– send commands or command loads to the satellite for on-board execution,
– oversee the transfer of the scientific data from the on-board tape recorders to ground systems for processing and analysis, and
– manage spacecraft resources (including on-board memory, batteries, and tape recorders).

To accomplish these activities, the analyst must thoroughly understand the operation of the spacecraft and ground systems and continuously monitor the current state of operations as indicated by telemetry parameters displayed on the POCC consoles. During an event, the analyst typically monitors hundreds of telemetry parameter values on multiple display pages that may be updating several times per second. Monitoring the operation of these satellites is a demanding, tedious task that requires well-trained individuals who are quick-thinking and composed under pressure.

As satellites become more complex, they become more difficult to operate. FOAs are reaching a level of information saturation as more and more data must be monitored and analyzed during real-time supports. Large volumes of low-level information can overwhelm analysts and disrupt their ability to identify and resolve conflicting constraints. Human operators may soon be unable to consistently monitor all of the information available. The need to automate some data monitoring functions is apparent.

Expert system technology is proving to be effective in automating some spacecraft monitoring functions. This paper first summarizes CLEAR, the first spacecraft monitoring expert system deployed at GSFC. The paper then reviews several lessons learned from CLEAR and other monitoring and fault isolation expert system projects undertaken at GSFC. Finally, the paper describes the Generic Spacecraft Analyst Assistant (GenSAA), a tool that will facilitate the development of future spacecraft monitoring expert systems. GenSAA has been defined based on the lessons learned from CLEAR and other expert system projects at GSFC.

## Initial Work: The CLEAR System

The Communications Link Expert Assistance Resource (CLEAR) is the first operational expert system at GSFC that automates one of the spacecraft analyst's tasks (Hughes & Hull, 1987). CLEAR is a fault-isolation expert system that supports real-time operations in the POCC for the Cosmic Background Explorer (COBE) mission. This system monitors the communications link between COBE and the Tracking and Data Relay Satellite (TDRS), alerts the analyst to any problems, and offers advice on how to correct them.

CLEAR is a forward chaining, rule-based system that operates in the COBE POCC. It monitors over 100 real-time performance parameters that represent the condition and operation of the spacecraft's communications with the relay satellite. Using this information, together with knowledge of TDRS operations, COBE's on-board communications system and the expected configuration of the scheduled event, CLEAR accurately portrays the status of the communications link.

Textual and graphical information about the condition of the COBE/TDRS/ground communications links is displayed in a tiled-window format. A graphics window displays the elements of the communications network from the COBE Spacecraft to the POCC; green lines represent healthy links between elements. When the performance parameters indicate that a communications link or processing system is degrading or down, the associated line or icon turns yellow or red, respectively. The display enables analysts to assess the current status of the communications event in a quick glance.

When CLEAR isolates a problem, a short description of the problem is displayed in a "problems" window. If multiple problems are found, the problem descriptions are ranked and displayed in descending order of criticality. CLEAR suggests analyst actions to correct the problem; however, the system does not take any corrective action itself.

To further assist the analyst and to provide support for its advice, the CLEAR system provides an explanation facility. When the analyst selects a problem displayed in the problems window, CLEAR provides a detailed explanation of why the expert system believes that the problem exists.

CLEAR has approximately 165 rules and isolates approximately 75 different problems. The types of problems include: non-reception of data within the control center (system or communication problems, or data reporting not activated); misconfigurations between the COBE POCC and the TDRS ground station (coherency/non-coherency, doppler compensation on/off, power mode, actual TDRS in use, antennae configurations); discrepancies in telemetry rate or format; inactive or non-locked links; and degrading or critical signal strength situations (Perkins & Truszkowski, 1990).

CLEAR operates on any of the seven PC/AT-class workstations that are used for console operations in the POCC. It is written in the 'C' language and uses the 'C' Language Integrated Production System (CLIPS) and a custom-developed graphics library.

The CLEAR Expert System has supported the COBE Flight Operations Team since launch in November 1989. It is used to monitor nearly all of the TDRS supports (COBE occasionally communicates directly to the Wallops ground station) and is regarded as the fault-isolation "expert" for the COBE/TDRS telecommunications link. CLEAR represents a successful attempt to automate a control center function using an expert system. Several other missions have requested to use it and, at the time of this writing, efforts are underway to adapt it to support the Gamma Ray Observatory mission which is scheduled for launch aboard the shuttle in Spring 1991.

## Lessons Learned

Several important lessons have been learned from the experience gained in developing and operating CLEAR. Key lessons have also been learned from other monitoring and fault isolation expert systems developed recently at GSFC, including the Ranging Equipment Diagnostic Expert System (REDEX) (Luczak, et. al., 1989), and other systems. These lessons learned have strongly influenced the definition of GenSAA.

• **Production rules effectively represent analysts' knowledge for automating fault-isolation in spacecraft operation.** The rule-based method of knowledge representation has proven to be quite powerful for fault-isolation in spacecraft operations. Production rules provide a direct method of encoding the fault-isolation knowledge of spacecraft analysts; the if-then structure closely parallels the stimulus-response behavior that they develop through extensive training. This knowledge can be translated smoothly into rule form. The development of CLEAR would have taken much longer using conventional, non-rule-based programming techniques.

• **Knowledge engineering is an iterative, time-consuming process.** Early in CLEAR's development, the primary concern was the perceived difficulty of the knowledge acquisition effort. However, the knowledge engineering task was found to be relatively straightforward, albeit time-consuming. The development of the rule base was a lengthy process due to the interactive nature of the knowledge acquisition. Basically, the expert would describe a specific piece of knowledge to the "knowledge engineer" who would attempt to transcribe it into a rule and pass it back to the expert for validation. When the rule accurately represented the piece of knowledge (which usually took multiple iterations between the expert and the knowledge engineer), it was passed to the test team for formal testing, and then, finally, released for operational use.

The involvement of various players in this process resulted in long turnaround times from the point at which a piece of knowledge was determined to be important until it was translated into a rule and placed into operation.

• **Allow the domain expert to participate in the rule formation.** The CLEAR development team learned that the knowledge structure of the fault-isolation process employed by the FOAs is shallow (i.e. the identification of a problem generally does not rely on the identification of other subproblems, and so on). Most of the problems identified by the analysts were discrete problems that seldom overlapped other problems. Conflicts between rules were minimal; this simplified testing, verification, and validation of the rulebase.

The participation of the analyst in knowledge acquisition and translation has many advantages. First, it can reduce the knowledge translation time and, more importantly, reduce knowledge translation errors that occur when a knowledge engineer formulates rules based on the knowledge extracted from documentation or interviews with the domain expert. Second, the verification and validation of the knowledge will be facilitated since the expert will better comprehend the rulebase. Third, the in-depth understanding of the knowledge base and its capabilities is likely to result in a higher degree of user confidence in the system thereby ensuring user acceptance.

• **Expect to fine-tune the expert system after it becomes operational.** For CLEAR, the rule-based method of knowledge representation has provided the flexibility to easily adapt the knowledge base to unforeseen changes in the operational behavior of the spacecraft. For example, even though the operational nature of COBE was fairly accurately understood by the design engineers and flight operations team before the launch, slight behavioral variations and complications arose once the spacecraft was in orbit. Although the FOAs were able to adjust to such variations quickly, some of the ground systems required complex software modifications. However, the changes required to CLEAR's rule-base were simple and quickly implemented. After modification, CLEAR provided consistent operational assistance. It is important to provide the capability to modify an operational expert system in a controlled, yet straightforward way.

• **Don't underestimate the integration process.** One of the most valuable lessons learned is that while prototypes can often be developed rapidly, operational expert systems require considerable effort. A major factor in this effort is the difficulty of interfacing the expert system to the data source.

The CLEAR development team learned that most of the development time for the operational system was spent on issues not directly related to the construction of the knowledge base. A surprising amount of effort focused on the integration of the expert system with the data source and graphics display system. This required in-depth programming knowledge of the interfacing systems and the ability to troubleshoot problems within them. Provide tools to simplify the complicated task of integrating the expert system with the interfacing systems and, if possible, reuse any interface code developed for a similar (expert) system.

• **Don't neglect the user-interface.** The human-computer interface is frequently the most underdeveloped component of an expert system. An effective user interface is inviting, comprehensible, credible, and simple to operate. To make it inviting, simplify the display layout and present only that information needed to efficiently perform the task. Graphics can greatly enhance the visual communications of a system; capitalize on their expressive power to provide system output that can be assimilated quickly and accurately.

The following lessons are also related to the use of graphical user interfaces:

- **Use colors prudently and consistently.** Although often misused, colors are valuable for emphasizing or coding information. Use them sparingly and in a manner consistent with other systems or conventions employed in the targeted operational environment.

- **Include a graphical user-interface (even in the first expert system prototype.)** CLEAR utilized a graphical user interface in the initial prototype to demonstrate the capabilities of the proposed expert system; this elicited valuable feedback from the expert and other FOAs. In contrast, a non-graphical user-interface was used in the initial REDEX prototype; as a result, user interest and feedback was limited early in the project

- **Use an object-oriented diagram editor to ease diagram creation and maintenance.** Ideally, the diagram editor should enable diagram objects to be easily associated with status values and fault conditions inferred by the expert system. In the REDEX project, a diagram editor with only limited capability was used, and as a result, significant effort was required to construct and modify diagrams.

• **Use block diagram displays to graphically illustrate the system being monitored.** Users have responded very positively to the use of schematic displays that graphically show monitored system status and fault locations. Analysts and technicians usually learn about the systems they monitor by studying system block diagrams in training classes and reference manuals. By using similar block diagram displays, a monitoring expert system can present status to the user in a familiar and intuitive format. Color coding of status conditions on such displays has been found to be an effective way to present succinct status summaries. For example, REDEX users have been enthusiastic about the color block and layout diagrams in that expert system; over 35 diagrams graphically depict the functional and physical structure of the equipment being diagnosed.

- **Make the system easy to operate.** Operation of the expert system should be simple enough that the user can concentrate on the problem, not on how to operate the system. The following techniques were applied in CLEAR and REDEX to simplify operation:

- **Reduce user input to a minimum.** CLEAR operates in a highly autonomous mode; no user input is required after system initialization. CLEAR has been well-accepted by its users, partially because it operates as an independent intelligent assistant, allowing the spacecraft analyst to focus on other responsibilities during real-time satellite contacts.

- **Use hypertext and hypergraphic techniques.** These techniques (Bielawski & Lewand, 1991) enable the expert system user to quickly select and display desired diagrams by clicking on link buttons that appear on each diagram. Links can be used to create diagram hierarchies, off-page connections, diagram sequences, and other structures. REDEX uses these techniques to enable users to navigate through a set of several dozen graphical display pages; they find the approach intuitive and easy to use.

These lessons learned have all influenced the definition and development of GenSAA.

## GenSAA

### *Overview*

GenSAA is an advanced tool that will enable spacecraft analysts to rapidly build simple real-time expert systems that perform spacecraft monitoring and fault isolation functions. Expert systems built using GenSAA will assist spacecraft analysts during real-time operations in spacecraft control centers. Use of GenSAA will reduce the development time and cost for new expert system applications in this domain. GenSAA will allow major expert system software functions and portions of knowledge bases to be reused from mission to mission.

GenSAA has the following primary characteristics:

- **Easily used** — The process for developing specific expert system applications using GenSAA will be straightforward enough that it can be performed by trained spacecraft analysts on the flight operations team.

- **Rule-based** — GenSAA will support the use of rules to represent spacecraft and payload monitoring and fault isolation knowledge. Rule-based representations are easily learned and can be used to describe many of the reasoning processes used by spacecraft analysts. (An object representation technique will be included in a subsequent phase.)

- **Highly graphical** — The GenSAA operational user interface will support both textual and graphical presentations of health and status information and fault isolation conclusions. Hypertext and hypergraphic techniques will be supported to simplify operational interaction with GenSAA expert systems.

- **Transparently interfaced with TPOCC** — GenSAA will be used to create expert system applications that will support analysts in spacecraft control centers that use the new Transportable Payload Operations Control Center (TPOCC) architecture. TPOCC is a new Unix-based control center system architecture that will be used on many new spacecraft missions at GSFC. GenSAA will be adaptable to also support non-TPOCC data interfaces.

GenSAA is being defined and prototyped by the Automation Technology section (Code 522.3) of the Data Systems Technology Division at GSFC. A system and operations concept has been defined (Hughes & Luczak, March 1990), and a multi-phase prototype effort is underway (Hughes & Luczak, August 1990).

### *GenSAA Architecture*

GenSAA is a shell, or software framework for developing spacecraft control center expert systems. It is analogous to many commercial expert system shells because it facilitates the development of specific expert system applications. However, GenSAA is tailored to the specific requirements of spacecraft analyst assistant expert systems in TPOCC control centers. GenSAA therefore shares many of TPOCC's architectural features.

The TPOCC architecture is based on distributed processing, industry standards, and commercial hardware and software components. It employs the client/server model of distributed processing, the Network File System (NFS) protocol for transparent network access to files, and the X Window System (X.11) with the Motif library and window manager for the graphical operator interface. A TPOCC configuration consists of a small set of specialized front-end processors and Unix-based workstations on an Ethernet network using the TCP/IP network protocol. GenSAA operates in this environment.

GenSAA allows spacecraft analysts to rapidly create simple expert systems without having to deal directly with the complicated details of the systems with which the expert system must interface. In addition, it will allow the expert system developer to utilize and/or modify previously developed rule bases and system components, thus facilitating software reuse and substantially reducing development time and effort.

Figure 1 shows the six major elements of GenSAA. They are divided into two sets: the GenSAA Workbench and the GenSAA Runtime Environment. These are described in the sections below.

### *The GenSAA Workbench*

The GenSAA Workbench is composed of three utilities that enable a spacecraft analyst to create a GenSAA application. A GenSAA application is a specific expert system that performs real-time monitoring and fault isolation functions in a TPOCC spacecraft control center.

A GenSAA application is created by defining the application's runtime specificiations using the GenSAA Workbench. Figure 2 illustrates that these specifications, called Reusable Application Components, together with the elements of the GenSAA Runtime Components, comprise a GenSAA Application.

The GenSAA Workbench utilities are as follows:

• **Data Interface Development utility** – This utility is used to create the Data Interface

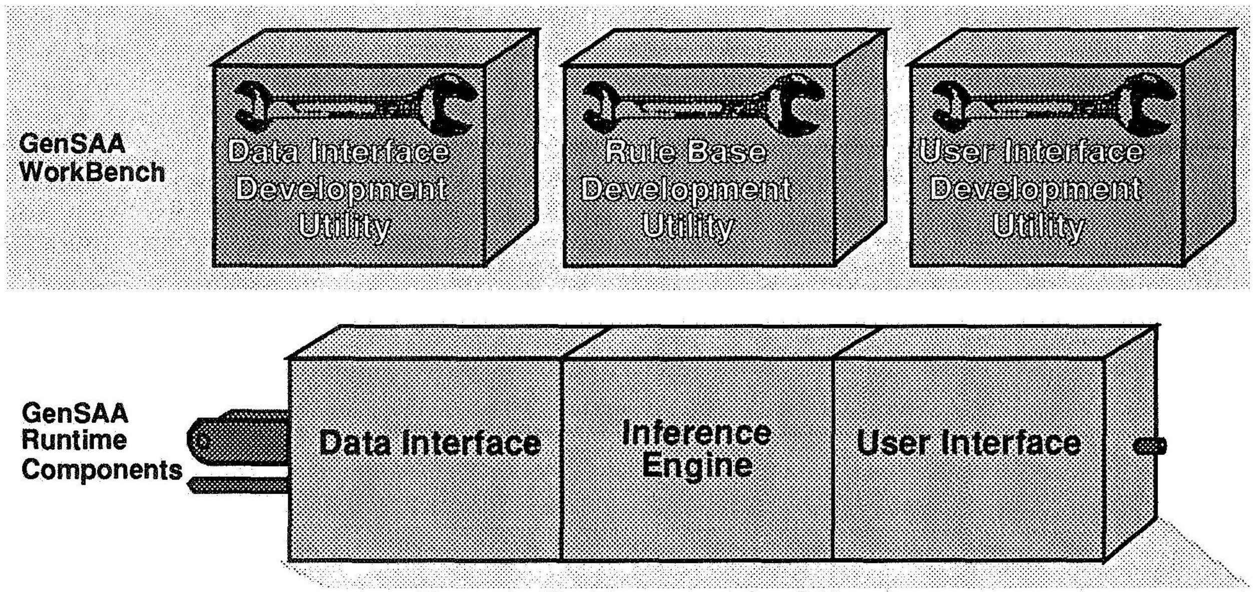

GenSAA WorkBench
Data Interface Development Utility
Rule Base Development Utility
User Interface Development Utility
GenSAA Runtime Components
Data Interface
Inference Engine
User Interface

Figure 1. The Elements of GenSAA

Specification for a GenSAA application. The Data Interface Specification defines three types of data that are used by the GenSAA application during real-time operations:

– *Telemetry data* – Telemetry data variables represent real-time status of the monitored spacecraft and related ground support systems. (Telemetry variables are sometimes called telemetry mnemonics.) Values for these variables are received and updated during spacecraft operation periods from the TPOCC Data Server process, which is part of the TPOCC software. Using the Data Interface Development Utility, the GenSAA Workbench user selects the telemetry variables needed for the expert system being created from a list of all the telemetry variables available from the TPOCC Data Server. Values for only those variables selected will be received by the expert system during run-time.

– *Configuration data* – Configuration data variables represent expected operating modes and equipment configurations. For example, a configuration variable might represent the setting of a switch that determines which of two redundant components is to be used. Values for these variables are entered by the spacecraft analyst during spacecraft operation periods.

– *Inferred data* – Inferred data variables represent conclusions inferred by rules in the rule base. For example, an inferred data variable might represent the health or fault status of a component in a spacecraft subsystem. (The name of an inferred data variable together with its current value is called an *inferred fact.*) Values are assigned to these variables by actions executed in the "then" part of rules that fire.

• **Rule Base Development utility** – This utility is used to create the rule base for a GenSAA application. The rule base is a set of expert system rules in "condition-action" ("if - then") format that may infer new facts based on currently asserted facts. The inference engine controls the firing of rules in the rule base during execution of the GenSAA application.

During run-time, if all the conditions of a rule are satisfied, then the rule fires and all its actions are performed. Conditions can be constructed using the telemetry, configuration, and inferred data variables specified with the Data Interface Development Utility. Actions may include: asserting/retracting an inferred fact, enabling/disabling a rule or ruleset, performing a mathematical calculation, and displaying text messages on the user interface.

• **User Interface Development utility** – This utility is used to create the User Interface Specification for a GenSAA application. The User Interface Specification defines the user interface panels and the layout and behavior of the display items that comprise the operational user interface of the GenSAA application. The Workbench user can create a variety of display items, including graphical icons, scrolling text lists, and data-driven objects such as meters, gauges, and simple strip charts. The display designer constructs a panel by dragging display items from a palette and placing them wherever desired. Lines can be drawn using connector items to create block diagram displays. The Workbench user can associate each display item with a telemetry, configuration, or inferred data variable, and specify how changes in the value of the variable will affect the presentation of the item. Characteristics of an item presentation that can change include its color, the icon displayed, and the position of the dynamic portion of a data-driven object. Simple drawing capabilities are provided to allow the creation of new graphical icons. Any display item can also be specified to be a hypertext button; clicking on such a button during run-time can cause an informational pop-up window to appear, or cause a new panel to be displayed.

The GenSAA Workbench utilities use a graphical, point-and-select method of interaction to facilitate use. The utilities are also highly inter-operable. For example, when using the Data Interface Development utility, the user may select a given telemetry mnemonic to be included in the Data Interface Specification. Later, when using the Rule Base Development utility, the user can easily access the Data Interface Specification and to reference the mnemonic in a condition of a rule. Similarly, when using the User Interface Development utility, the user can again easily access the Data Interface Specification when associating telemetry mnemonics with display objects.

In Figure 2, the outputs of the GenSAA Workbench utilities are described as reusable application components. These specifications will be placed in a library so that they can be reused in creating the specifications for new GenSAA applications. Operations like cut and paste will be available to allow portions of previously created specifications to be used in constructing a new specification.

### *GenSAA Runtime Environment*

The elements of the GenSAA Runtime Environment are called the GenSAA Runtime Components; they are used without change in each GenSAA application. They control the operation of a GenSAA application during its execution in a TPOCC control center. They read the Data Interface Specification, Rule Base, and User Interface Specification files to determine the specific behavior of the GenSAA application. Each of the GenSAA Runtime Components is implemented as a separate Unix process; they communicate with one another via shared memory and message queues. Their functions are as follows:

- **Data Interface** – This component requests telemetry from the TPOCC Data Server, as specified in the Data Interface Specification. It reformats the real-time data it receives and makes it available to the Inference Engine and User Interface components. (It also exchanges data with the GenSAA Data Server, as described below in the section Multiple GenSAA Applications.)

- **Inference Engine** – This component controls the firing of rules in the rule base. A rule is fired when all its conditions are satisfied; the conditions will often involve the current values of telemetry, configuration, and inferred data variables. Inferred facts and messages may be sent to the User Interface component and displayed to the spacecraft analyst as defined in the User Interface Specification. NASA's CLIPS inference engine forms the core of this component.

- **User Interface** – This component manages the user interface of the GenSAA Application. It displays user interface panels that contain both text and graphics. Color is used to enhance the display of state data. Data-driven display objects are associated with telemetry values received from the TPOCC data server and

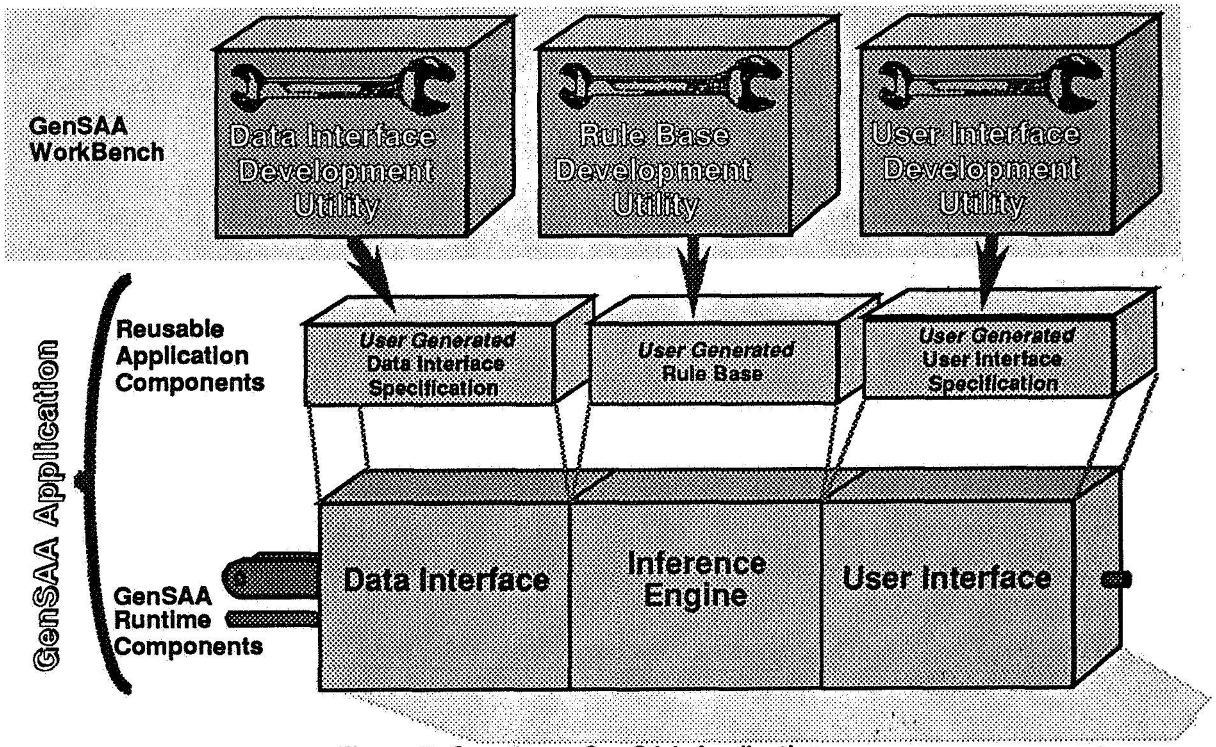

GenSAA WorkBench
Data Interface Development Utility
Rule Base Development Utility
User Interface Development Utility
GenSAA Application
Reusable Application Components
User Generated Data Interface Specification
User Generated Rule Base
User Generated User Interface Specification
GenSAA Runtime Components
Data Interface
Inference Engine
User Interface

Figure 2. Creating a GenSAA Application

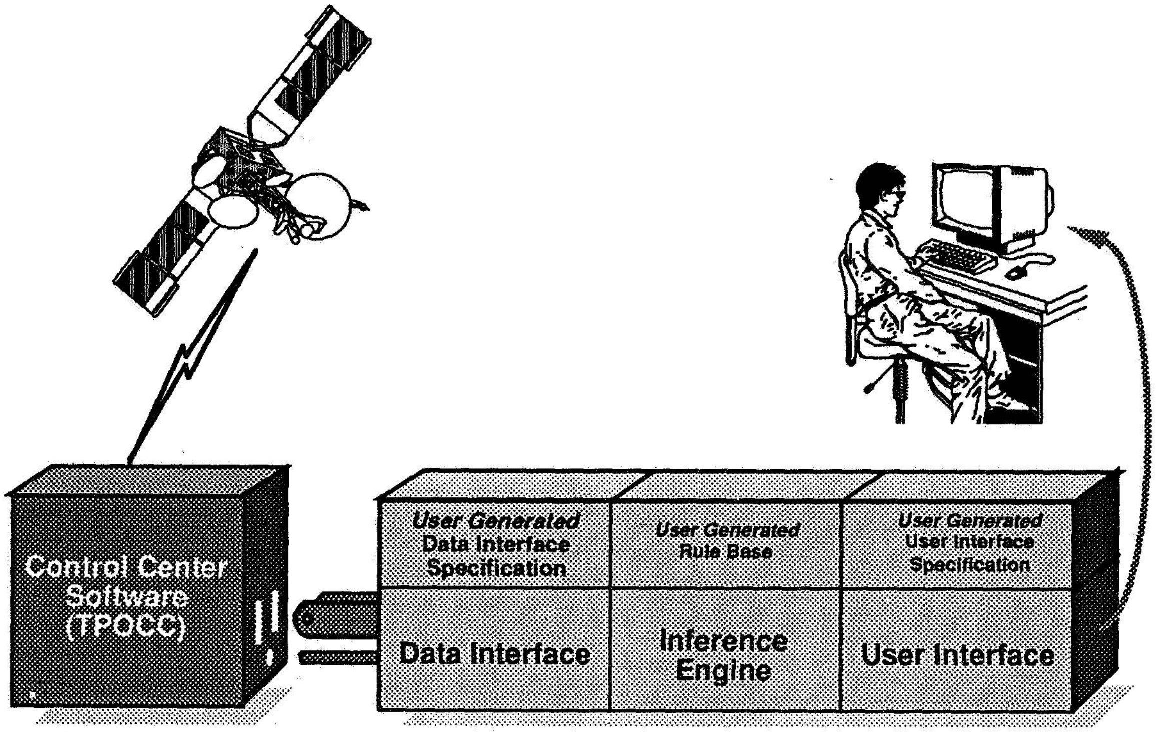

Control Center Software (TPOCC)
User Generated Data Interface Specification
User Generated Rule Base
User Generated User Interface Specification
Data Interface
Inference Engine
User Interface

Figure 3. A GenSAA Expert System Application in operation

inferred facts and conclusions received from the Inference Engine. In response to user inputs that include hypertext button events, the User Interface displays selected display panels, help text, and other informational text. The user interface panels, data-driven objects, and interaction objects are defined in the User Interface Specification that was generated by the GenSAA User Interface Development utility.

Figure 3 shows a completed GenSAA expert system application in operation. GenSAA expert systems will run on Unix workstations using the X Window System. The operational interface with the spacecraft analyst will typically include color block diagrams and animated data-driven objects (such as rotating meters, sliding bar graphs, and toggle switches) that graphically display the dynamic values of telemetry data, configuration data, and inferred conditions. The user interface will also typically contain hypertext and hypergraphic links to make it easy for the spacecraft analyst to quickly select desired display panels. The GenSAA Workbench supports the creation of these interface features.

### *Multiple GenSAA Applications*

GenSAA applications are intended to be relatively simple expert systems with small rule bases that are typically developed by a single analyst. For example, a GenSAA application might monitor and isolate faults for one subsystem on board a spacecraft. To handle more complex monitoring situations, involving for example several spacecraft subsystems, multiple GenSAA applications can be built and executed concurrently. Each GenSAA application would be allocated one portion of the monitoring task, and share key conclusions with one another.

A fourth component of the GenSAA Runtime Environment, the GenSAA Data Server, is used to enable multiple GenSAA applications to exchange data. As shown in Figure 4, the GenSAA Data Server is a Unix process that can receive a real-time stream of configuration and inferred data variable updates from any GenSAA application. The GenSAA Data Server distributes the data to any GenSAA application that has requested it. A given GenSAA

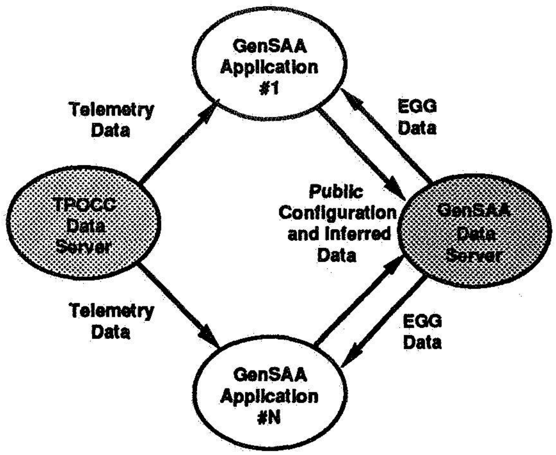


Figure 4. Sharing data among multiple GenSAA Applications

application only receives those variables it has specifically requested. The data received by a GenSAA application from the GenSAA Data Server is called externally generated GenSAA (EGG) data. A GenSAA application receives EGG data via its Data Interface component in exactly the same way as it receives telemetry data from the TPOCC Data Server.

Within a GenSAA application, EGG data can be used in the conditions of rules, and can be associated with display items in exactly the same way as telemetry, configuration, and inferred data. The Workbench supports the specification of EGG data as a fourth variable type. The Workbench also allows any local configuration or inferred data to be specified as public, to cause it to be sent to the GenSAA Data Server, and thereby shared with other GenSAA applications.

### *Benefits of GenSAA*

The following benefits are expected to be realized by using GenSAA to build spacecraft monitoring expert systems for future NASA missions:

• **Assists the FOAs with data monitoring**— FOAs monitor real time data looking for combinations of telemetry parameter values, trends, and other indications that may signify a problem with the satellite or its instruments. The expert systems created with GenSAA will assist the FOAs with the tedious task of data monitoring and allow them to focus on other, higher-level responsibilities during real-time contacts with the satellite. This, in turn, will likely result in more efficient and effective operations.

• **Reduces development time and effort; allows quicker response to necessary modifications** — The behavior of an orbiting satellite is quite dynamic and occasionally different than anticipated. To quickly create or modify expert systems that can effectively monitor satellites, tools are needed that allow analysts to formulate rulebases easily without the intervention or delay of knowledge engineers and programmers. Several benefits are expected by eliminating these traditional developers. Analysts will be able to create rules quickly in response to unforeseen changes in spacecraft behavior or operational procedures. Also, knowledge translation errors will be reduced or, at least, more easily corrected. Knowledge translation errors are errors which are inadvertently introduced during the process of translating a piece of expert knowledge into rule form.

• **Serves as a training tool** — In addition to assisting the FOAs with real-time spacecraft operations, GenSAA will be useful as a training tool in two ways. First, by utilizing the playback utilities provided by TPOCC, analysts will be able to replay a previous spacecraft communications event. Thus, a student analyst can observe how the expert system handles a specific problem scenario. Exercises like this will provide a realistic, hands-on environment for training FOAs in a safe, off-line mode. Second, experience from previous expert system projects indicates that the development of rules used in an expert system is a beneficial mental training exercise for the FOA. When FOAs create rules themselves, they must consider alternatives more closely and may therefore develop a deeper understanding of the problem domain. This approach may enable more effective fault isolation methods to be identified.

• **Protects against loss of expertise** — Another benefit of automating fault-isolation tasks with rule-based systems is that the resulting rulebase serves as accurate documentation of the fault-isolation method. The rulebase can be studied by student analysts to learn about fault-isolation techniques. Even more importantly, mission operations can be better protected against the effects of personnel turnovers. POCC expert systems that capture fault-isolation knowledge preserve expertise from mission to mission and mitigate the impact of the loss of experienced FOAs.

GenSAA is well suited for use on spacecraft projects that involve a series of similar but nonidentical missions such as NASA's Small Explorer (SMEX) and International Solar-Terrestrial Physics (ISTP) programs.

## Conclusion

As satellites become more complex, their operation is becoming increasingly difficult. FOAs who are responsible for the command, control, health, and safety of these spacecraft must monitor increasing volumes of data, and are quickly reaching a level of information saturation. As demonstrated by the CLEAR Expert System, fault-isolation expert systems can help FOAs monitor the flood of data. Expert systems can accurately monitor hundreds of real-time telemetry parameters, isolate discrepancies and anomalies the instant they can be detected, and alert the analysts and provide advice on how to correct problems swiftly and effectively. Unfortunately, development of these systems is often time consuming and costly moreover, they often cannot be easily reused for other missions.

Consequently, GenSAA is being developed for use by the FOAs who work in satellite control centers. GenSAA is designed to enable fault-isolation expert systems to be developed quickly and easily, and without the delay or costs of knowledge engineers and programmers. By facilitating the reuse of expert system elements from mission to mission, GenSAA will reduce development costs, preserve expertise between missions and during periods of personnel turnover, and provide more effective spacecraft monitoring capabilities on future missions.

## References

Bielawski, L., & Lewand, R. (1991). Intelligent Systems Design: Integrating Expert Systems, Hypermedia, and Database Technologies, New York: John Wiley & Sons.

Hughes, P.M. (1989). Integrating Expert Systems into an Operational Environment. AIAA Computers in Aerospace VII Conference, Monterey, California.

Hughes, P.M., & Hull, L.G. (1987, May). CLEAR: Communications Link Expert Assistance Resource. 1987 Goddard Conference on Space Applications of Artificial Intelligence and Robotics, Greenbelt, Maryland.

Hughes, P.M., & Luczak, E. (1990, March). Generic Spacecraft Analyst Assistant-Concept Definition. Data Systems Technology Laboratory Document DSTL-90-008. Greenbelt, Maryland: NASA/Goddard Space Flight Center.

Hughes, P.M., & Luczak, E. (1990, August). Generic Spacecraft Analyst Assistant-Prototype Definition. Data Systems Technology Laboratory Document DSTL-90-033. Greenbelt, Maryland: NASA/Goddard Space Flight Center.

Luczak, E.C., Gopalakrishnan, K., & Zillig, D.J. (1989, May). REDEX: The Ranging Equipment Diagnostic Expert System. 1989 Goddard Conference on Space Applications of Artificial Intelligence, Greenbelt, Maryland.

Perkins, D., & Truszkowski, W. (1990). Launching AI in NASA Ground Systems. AIAA/NASA Second International Symposium on Space Information Systems, Pasedena, California.

N91-22780

# Representing Functions/Procedures and Processes/Structures for Analysis of Effects of Failures on Functions and Operations

Jane T. Malin
Automation and Robotics Division - ER2
NASA Lyndon B. Johnson Space Center
Houston, Texas 77058

Daniel B. Leifker
The MITRE Corporation
1120 NASA Road One
Houston, Texas 77058

## *Abstract*

Current qualitative device and process models represent only the structure and behavior of physical systems. However, systems in the real world include goal-oriented activities that generally cannot be easily represented using current modeling techniques. We propose an extension of a qualitative modeling system, known as *functional modeling*, which captures goal-oriented activities explicitly, and we show how they may be used to support intelligent automation and fault management.

## *Background*

Artificial intelligence (AI) technology for intelligent automation of monitoring, control, and fault management of space systems will result in significant reductions in operational costs of manned and unmanned systems, as well as increased capability to carry out new types of unmanned missions. In addition, more robust fault management performance will reduce costs for space system maintenance and repair and can potentially reduce risks from undetected failures.

An important goal of work in AI is to produce software that can respond constructively to a wide class of problem scenarios. At the same time, however, the software should operate in ways that reflect human thought and reasoning patterns. Representations have therefore been developed that either correspond to or mesh with conventional human perspectives of the problem domain. Understandable rule-based and object-based systems have been successfully developed and used by flight controllers in the Space Shuttle program (Muratore, 1990).

Additional types of representations are needed to capture key concepts and strategies that are used for fault management by mission controllers, system designers, and safety personnel. They use mental models of the function and structure of designed systems and their interrelated components. These models require more expressive power and an enlarged scope before they can be added to the set of AI representations that are now successfully used.

## *Modeling Structure, Behavior, and Function*

Computer simulations and models are used to represent, to any desired level of detail, the structures and actions of physical systems. Modeling is done primarily to understand certain dynamic principles related to the given system that cannot be analyzed in closed form and cannot be studied in the system itself without great cost, danger, or inconvenience.

A number of researchers have developed a rich simulation theory known as qualitative modeling (see, for example, Forbus, 1984; Davis and Hamscher, 1988). The concept we present here, functional modeling, is an outgrowth of recent attempts to merge qualitative modeling and discrete event simulation into a single unified paradigm. Malin, Basham, and Harris have implemented this paradigm in a system known as CONFIG (Malin et al., 1990), a modeling and simulation tool prototype for analyzing normal and faulty qualitative behaviors of engineered systems. Like other device modeling systems, CONFIG has relied on component structures and processes to represent real-world systems.

However, current structure and process approaches rarely provide a way to represent system functionality. In fact, there have been admonitions to not mix function into a model of system structure and behavior. Nevertheless, since devices are systems designed to be used in goal-oriented activity, functionality is important to model and analyze. Modelers need representations of functionality to evaluate the success of a design, that is, to analyze how well the device will perform its functions. We argue that the "functionality" of the device is a concept that captures how activities (the operations/acts of the device/structure in time) produce a set of effects that are goals the device is designed to achieve. The structure of the device is a set of components and their interrelationships. The behavior of the device consists of what it does, and can include internal processes that cause changes in itself or in things it operates on. Thus, to model functionality, one must model structure, behaviors, goals and time -- the principal constituents of a goal-oriented activity. Since procedures are used by human and machine controllers to sequence and structure goal-oriented activities, representing function also provides the basis for representing procedures and controllers. Goals reside in controllers and in designers of devices and procedures, and are not ordinarily part of the physical device itself. Nevertheless, they must be combined with representations of device structure and behavior to model and analyze the functionality of the design.

The Space Shuttle Remote Manipulator System (RMS) can be used to illustrate this. Its structure, as shown in Figure 1, is made up of components such as the manipulator arm, manipulator retention latches (MRL), manipulator positioning mechanisms (MPM), and payload capturing subsystems. Its behavior consists of various movements and capture and release operations. The system has been designed to perform functions such as deploying a payload while avoiding collisions and remaining operational. These serve goals of safety and transport.

Clearly, fault management systems can be made more useful if they embody models that represent the entire scope and range of the monitored physical systems. Such a system for the RMS, for example, would be totally integrated with RMS procedures (which, being goal-oriented, are not a part of RMS designed structure and behavior) and thus support RMS fault management activities at new levels of operation. To make these ideas more precise, we turn now to a detailed discussion of real-world systems and how they may be modeled in terms of function as well as behavior.

### *Functional Systems*

Figure 2 depicts the organization of a real-world designed system and its relationship to real-world goals. It is tempting at first to view the *entire* designed system as a physical device with a given structure and behavior. Using the RMS example, however, it is easy to see that this is not the case: the entire RMS designed system encompasses not only the physical device but also a collection of related procedures that govern the (presumably judicious) use of the RMS device to achieve goals. A procedure is an information structure whose purpose is to help a controller successfully carry out and monitor plans. A procedure also contains information to support impact assessment or even fault management replanning and recovery.

The bridge between the procedure and the device is the controller, which (a) consults the procedure and decides on an action, (b) determines the true state of the device by requesting specific monitoring information,

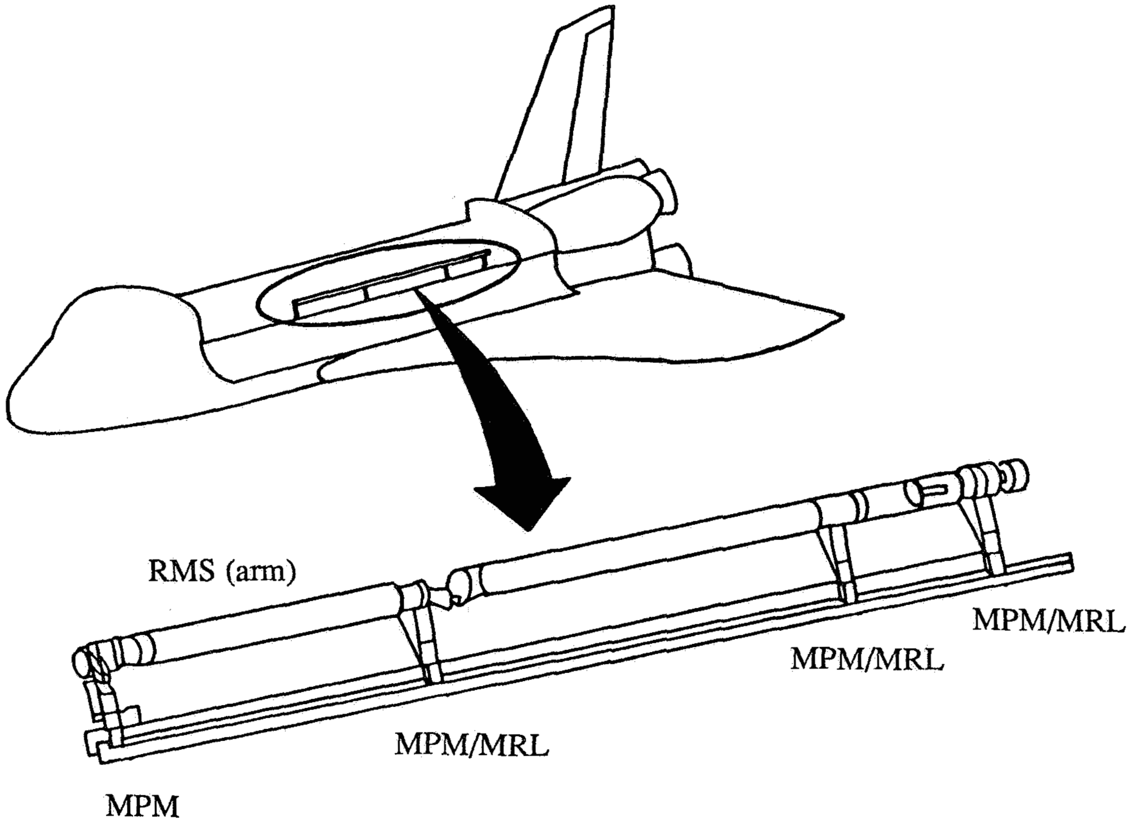


Figure 1: Manipulator Arm

(c) decides if the true state of the device is appropriate for the desired action, (d) executes the action by issuing appropriate commands to the device, and (e) confirms success of the action. This is the normal sequence for the careful monitoring of physical systems, and we will revisit this sequence later. In the meantime, we use the term *functional system* to refer to the combination of a designed system and its goals in the real world.

Several things should be noted from Figure 2. First, the notion of controller of quite general; in the real world it could be a human or an expert system. Second, we use the term *procedure*[1] rather loosely, and it should not be confused with or compared to an algorithm. As will be described later, procedures are realized as networks of goal-oriented activities that have no counterparts in conventional algorithms. Third, the goals of the functional system are not a part of the designed system. Some goals may be clearly implied by the procedures of a designed system, but they have a separate and detached existence. There is obviously a close relationship between the two in the sense that one can, to some extent, imply or describe the other, but they are distinct entities. In fact, two different designed systems may have identical goals.

### *Behavioral Models*

As shown in Figure 3, most device models contain interconnected *components* that simulate the structure of a target device. For example, an RMS latch is modeled by a software "latch" whose possible operational

[1] Terms such as "Shuttle Operations Procedure" illustrate our usage of *procedure*.

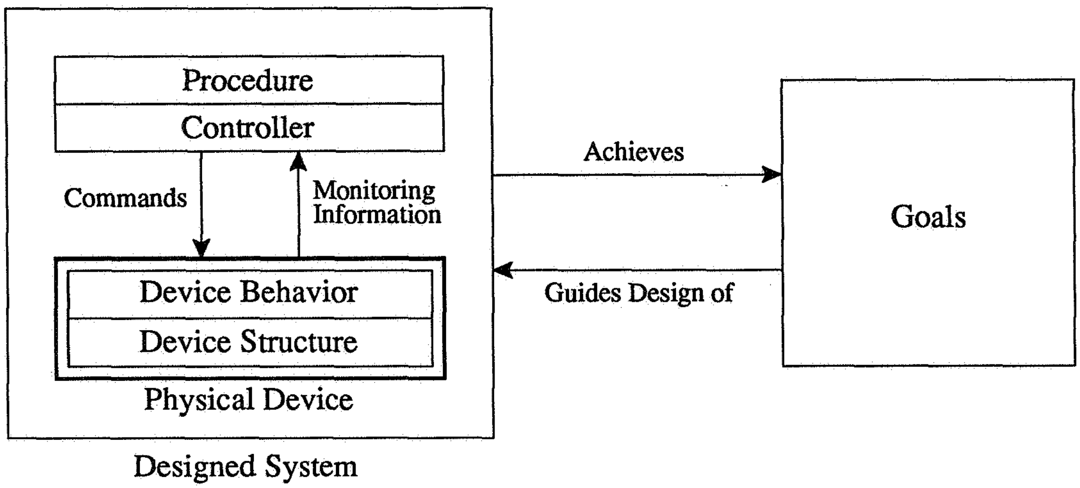


Figure 2: Real-World Functional System

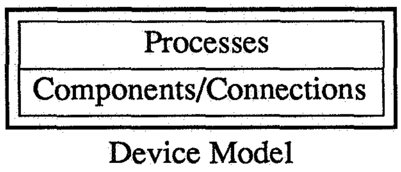


Figure 3: Behavioral Model

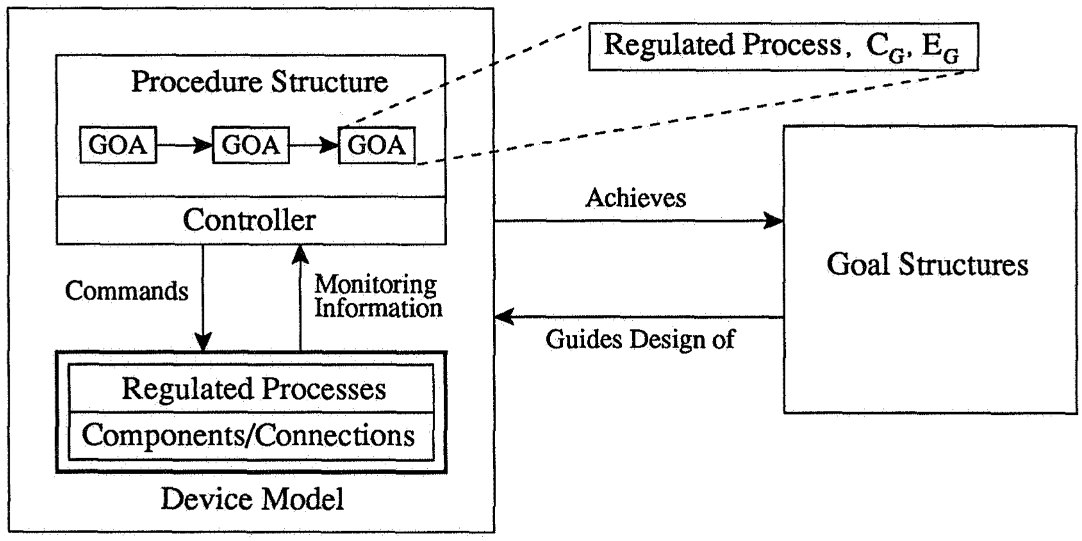


Figure 4: Functional Model

modes mimic the modes of the physical latch (open, closed, or in transition). Hence, during simulation, a component must have some sort of memory to distinguish its current mode from its potential modes. Such a memory is typically implemented as a collection of *state variables*. They may be used to remember not only component modes but also any relevant information that supports the purpose of the simulation. Conventional models could manage state variables that represent things such as temperatures, pressures, rates, oscillations, and even acceleration.

Similarly, the behavior of the target device is represented by *processes*. Each component has associated with it a set of processes that simulate device dynamics by changing the values of state variables in the model. A change in a state variable may activate one or more processes, which themselves can change other state variables. In this fashion the entire structure and behavior of a device can be simulated efficiently. Of course, depending on the exact combinations of state variable values, a model may rest in any one of a very large number of possible states. However, most of these states can be grouped into configurations that denote the general condition of the system (e.g., deployed, not deployed, nominal, off-nominal, etc).

Such models, which we call *behavioral models*, only simulate the device portion of the functional system and thus ignore the issues of commanded behavior, goal-oriented activities, procedures, and goals. Having no explicit links to outside regulation or intention, the behavioral model is primarily used to generate potential outcomes that would occur by starting the device from a given state. It can be started, halted, analyzed in detail, and then restarted as often as desired, but, in the absence of constant manual intervention, it remains largely insulated from outside influences. Thus users of behavioral models are more observers than controllers.

CONFIG is an example of a behavioral model. Qualitative component models are defined in terms of normal and faulty modes and processes, which are defined by invocation statements and effect statements with time delays. System models are contructed graphically by using instances of components and relations from object-oriented model libraries. System failure syndromes are simulated in CONFIG using a modified form of discrete event simulation. Yet despite these capabilities, CONFIG only models device structure and behavior. Extending CONFIG to model goals as well is the chief purpose of our research into functional modeling.

### *Functional Models*

Figure 4 shows how an entire functional system may be modeled by what we term a *functional model.* Corresponding to the physical device is a device model with components and processes, just as with behavioral models. In addition, however, the functional model contains *procedure structures* and *goal structures* that correspond to procedures and goals in the real world. A controller executes the procedure structure on the device model. Goal structures are important for modeling complete functional systems. However, since they are separate from the designed system, goal structures will not be considered further in this paper.

There is an important difference between a functional device model and a behavioral device model: the functional device model is constantly interacting with the controller. Explicit connections are required to pass commands from the controller to the device model and to transmit data from the model to the controller for use in monitoring.

Hence processes in functional device models are extensions of processes in behavioral device models. We call the resulting extension a *regulated process*. The functional model also contains a *procedure structure* to model procedures in the functional system. These procedures consist of networks of *goal-oriented activities*. These three concepts are key features of functional modeling, and each will now be discussed in detail.

### *Regulated Processes*

Regulated process are models of device behavior in real-world functional systems. They are analogous to ordinary processes in behavioral models such as CONFIG except that they have explicit mechanisms for communicating with controllers. This communication may be from the controller to the regulated process (commands) or from the regulated process to the controller (control information). This section will describe these mechanisms in detail.

The general layout of a regulated process is given in Figure 5. There four parts: the regulator, the invocation, the outcome, and the effector.

*Regulator* (command from controller)
- Start Switch
  - Inhibit On (default)
  - Inhibit Off
- Terminate Switch
  - Halt at Completion (default)
  - Abort Immediately

*Invocation* (conjunction of conditions)
- Component (self)
  - Mode
  - State variable values
- System
  - Other system components and their modes
  - Other system state variable values
  - Component connections
- Subsystem
  - Subcomponent and their modes
  - Submodel state variable value
  - Subcomponent connections
- Supersystem
  - Supercomponent modes
  - Supermodel state variable values
  - Supercomponent connections

*Outcome* (list of effects)
- Results of process (changes to mode and state variables)
- Delay

*Effector* (means of achieving effects)
- Pointer to procedure in submodel

Figure 5: The Regulated Process

The *regulator* may be regarded as two optional switches that control when the process starts and terminates.

The start switch, if present, has two modes: inhibit on and inhibit off. The regulated process may not begin execution if the start switch is inhibit on. The default mode is inhibit on. There are thus two scenarios for initiating a regulated process: (1) there is no start switch present, in which case the regulated process proceeds normally whenever the simulation triggers it; and (2) there is a start switch, in which case the regulated process emerges in the simulation in a "waiting" state, and the controller explicitly commands a "start!" by toggling the start switch from inhibit on to inhibit off. (Switching back to inhibit on at this point has no effect whatsoever on the running process.)

The terminate switch, if present, also has two modes: halt at completion and abort immediately. The default is halt at completion. The two scenarios for use are as follows: (1) there is no terminate switch present, in which case the process runs as expected and terminates when its outcome is completed; and (2) there is a terminate switch, in which case the process halts as soon as the controller an abort (or runs until expected completion if the controller never switches to abort).

The *invocation* is a conjunction of state expressions[2] required for the process to begin. An invocation (denoted C) may be formally decomposed as ($c_1$ AND $c_2$ AND ... AND $c_n$) where $c_i$ is a state expression called a *condition*. It is stressed that conditions merely denote the requirements necessary

[2] Device models, both in behavioral and functional models, rely on state variables. It is assumed that these variables are visible throughout the entire system and that their values may be examined and tested at any time. We define a *state expression* as any Boolean expression built from these state variables. The only requirement for state expressions is that they be well-defined (i.e., ultimately evaluate to true or false) at all times. Thus state expressions may be as simple as a comparision between a state variable and some value, or it may be a very complex expression involving many of these comparisons or nested subexpressions joined by Boolean operators.

for the process to be physically executed and they imply nothing else. For example, the MRL process "release arm" would have as part of its invocation a condition that the motors are operating nominally, and a condition that the terminating microswitches are operational, among other things. For convenience, these conditions are grouped according to their "location" within the model hierarchy. A given regulated process is associated with a specific component. Conditions using the modes and state variables of that component are given first, followed by conditions using modes, connections, and state variables of other components at the current level within the model. But some conditions may also use modes and state variables from a subsystem or a supersystem of this component, and they appear last. This grouping, or "invocation typing," is merely a form of semantic clustering and is done solely to mirror human perspectives.

The *outcome* is a list of state expressions which become true during the process or when the process terminates. An outcome (denoted E) may be formally decomposed as $(e_1{:}d_1, e_2{:}d_2, e_3{:}d_3, \ldots, e_n{:}d_n)$ where $e_i$ is a state expression called an *effect* and $d_i$ is the delay, or the time lapse between the start of the process and the point when the effect becomes true. Effects are counterparts to conditions in the invocation. However, instead of specifying physical requirements for process execution, they represent expected states of the model that result during or after process execution.

The *effector*, another optional part of the regulated process, names the procedure(s) in the submodel which actually implement the outcome. A given component *S* may consist of *n* subcomponents $s_1, s_2, \ldots, s_n$. A regulated process *R* associated with *S* achieves its outcome simply by resetting *S*'s state variables and ignoring *S*'s subcomponents. However, the controller may wish to explicitly simulate the activities of all subcomponents $s_i$ and not just assume that they will produce the outcome in *R*. In this case, a procedure must be created at the submodel level (the $s_i$ level) to achieve these goals explicitly. Whenever this occurs, that procedure is named by the effector of *R*.

Hence controllers may issue commands to the regulated process by setting the regulator switches, and they may retrieve control information from the regulated process simply by referencing the appropriate state variables.

### *Goal-Oriented Activities*

As controllers, humans can never monitor every facet of a real-world functional system at every moment. We are forced to monitor only a subset of the system and simply to make assumptions about the unmonitored parts. Unfortunately, a real-world functional system is not obligated to obey our assumptions, and disasters can occur whenever it does not. Hence good controllers are fundamentally suspicious when following plans. They do not blindly execute the specified steps, but instead will view each step as a separate goal with distinct conditions and expected effects. Furthermore, whenever possible, they will independently prove selected conditions and confirm the effects even if the system gives no hint of a failure.

Goals such as these must be reduced to formal objects before they can be manipulated by computer-based systems. We next show how goals and regulated processes are related.

Balkanski (Balkanski, 1990) defines *activity* as a triple containing an act-type, an agent which performs the act-type, and the time interval over which the act-type is performed. This formalism is useful in modeling collaborative activities, but it also coincides with regulated processes in the sense that a regulated process embodies an activity of the type Balkanski describes. We extend this notion to capture the concepts of monitoring and fault management. A *goal-oriented activity (GOA) with respect to device model M* is a a triple $(R, C_G, E_G)$ where

- R is a regulated process in M (with an invocation C and an outcome E)

- $C_G$, the *goal conditions*, is a conjunction of conditions contained in C

- $E_G$, the *goal effects*, is a list of effects contained in E

Note that $C_G$ may equal C, it may be a "subset" of C, or it may be null. A similar relationship holds between $E_G$ and E.

An example helps to clarify these ideas. As will be described later, the designer of a procedure usually expresses the steps of the procedure as distinct GOAs and not as simple commands. The device model contains a library of regulated processes[3], each with preset invocations and outcomes. Using the outcomes as a guide, the designer selects a regulated process which best accomplishes the intended purpose. It is important to note that a given regulated process may have many effects in its outcome, but not all of them may be intended effects of the designed procedure. The designer must therefore select a set of intended effects, $E_G$, which deserve special attention during procedure execution.

Although all the conditions must be true to begin execution of the regulated process, the designer may not wish to "suspect" them all for monitoring and fault management purposes. Analogously, the designer constructs a set of critical "contended" conditions, $C_G$, which must be reconfirmed as true (even if the model gives no indication to the contrary) before the regulated process is started.

Thus a goal-oriented activity is a "cross section" of the invocation and outcome of a selected regulated process. More precisely, a GOA is simply a view of a regulated process. This feature lets the system designer create general regulated processes that can be customized by a GOA in a procedure.

After the system has been designed, the controller encounters a GOA while using the system (i.e., executing a designed procedure). The controller notes the contended conditions $C_G$ and takes steps to confirm their truth before attempting to initiate the regulated process. If any conditions in $C_G$ are proved false, then there is a discrepancy between the controller's assumptions and the true state of the world. At this point, executing the regulated process would fail. A comparison with behavioral models will illustrate this critical concept. In a behavioral model, the process simply fails with no indication of why. In a functional model, the failure is likely to be intercepted before it occurs, and the controller, having tested $C_G$ explicitly, now enjoys considerable insight into the reasons for the averted failure.

### *Procedure Structures*

We have shown that procedures are used to achieve goals using devices whose current states are only assumed by the controller. For this reason, procedures cannot be expressed in terms of conventional algorithms; they must be expressed in terms of more abstract goal-oriented activities. This section describes the methods for constructing procedures out of GOAs.

The most straightforward approach is to implement the procedure as a sequence of independent GOAs. The controller is never forced to choose which GOA to execute next, and, furthermore, each GOA can ignore the results of previous GOAs in the procedures. This may suffice for simple procedures, but in general it seems clear that (1) procedures must have a memory, and (2) mechanisms must exist for combining GOAs in complex ways.

The first requirement may be satisfied easily by introducing data stores, called *procedure variables*, whose purpose is identical to variables in conventional computer programs. They may be set, reset, and

[3] Strictly speaking, a process with neither a start switch nor a termination switch is not truly regulated, but we still refer to them as "regulated" because the switches can be installed at any time.

tested by the procedure. Their purpose is simply to remember whatever the procedure chooses to remember, and they assist the controller by making available the results of previous steps in the execution of the procedure.

The second requirement is more difficult. We adapt Kant's proposals (Kant, 1988) for algorithmic control of goal-like entities. This approach allows GOAs to be chained in *control networks* that support classical programming control structures such as branching and iteration.

The general layout of a procedure structure is given in Figure 6.

*Preamble*
- Processes that point to this procedure
- Required resources
- Time estimate
- Global preconditions
- Procedure variable declarations
- Names of GOAs

*Results*
- Intermediate effects (temporary)
  - Primary
  - Side
- Final effects (permanent)
  - Primary
  - Side

*Control*
- GOA control network

Figure 6: Procedure Structure

Procedures consist of three parts. The *preamble* contains documentation information such as: the names of the regulated processes (presumably from one level higher in the model) that point to this procedure, a summary estimate of resources required to execute the procedure (e.g., fuel), a summary estimate of how long the procedure takes to execute, a set of global preconditions that describe the required general state of the system before the procedure may be executed, the declarations for procedure variables, and the names of all goal-oriented activities contained in the procedure. The *results* document the effects of the procedure and are classified by time (intermediate or final) and by intention (desired effect or side effect). Finally, the *control* section contains the control network of GOAs described above.

Having described regulated processes, goal-oriented activities, and procedure, we now present an example of how they might be used.

### *Sample RMS Application*

Preparation of the Shuttle RMS arm for mission tasks involves deploying the arm away from the Shuttle Payload Bay to its operational position. This includes the RMS Powerup and Deploy procedure, which uses the Manipulator Positioning Mechanisms (MPM) to swing the arm outboard from the Shuttle, uses the Manipulator Retention Latches (MRL) to unlatch the arm from the MPM, and then uses normal joint-driving commands to move the arm away from the MPM and its latches.

The first portion of this procedure consists of the following sequence of goal-oriented activities:

1. Select RMS: Supply power to the RMS control sensors and actuators
2. Configure power: Supply power to the RMS control sensors and actuators
3. MPM Deploy
4. MRL Release
5. Configure Power: Deactivate power supply to MPM/MRL motors

A more detailed representation of the MRL Release GOA (number 4) is provided. This unlatching process includes the following detail:

1. Check whether the MRL drive motors are operational
2. Check that the Shuttle Digital Autopilot is in free-drift or the vernier navigation jets are selected
3. Start the MRL release of the latches

|  |  | Regulating Command |  |
|---|---|---|---|
|  |  | Start | Abort |
| Regulating Information | Goal Conditions | Subsystem<br>At least one operational drive motor per latch<br>Shuttle<br>Digital autopilot mode free drift or vernier jet selected | Subsystem<br>NOT (At least one operational drive motor per latch)<br>Shuttle<br>NOT (Digital autopilot mode free drift or vernier jet selected) |
|  | Goal Effects | Self<br>NOT (MRL Released) | Self<br>MRL released |

Figure 7: Controller commands for the MRL Release Process

4. Monitor the MRL release, and abort the release if it has not terminated properly after 18 seconds

This procedure step exhibits much of the detail that the GOA view of a regulated process is designed to capture. There are two types of regulation commands used: start! (set inhibit off) and abort! (set abort switch on).

Figure 7 shows how information about selected conditions and effects of the MRL Release process determines whether each command should be issued by the controller. This information is at various levels of detail within the shuttle system, from the operational status of motors that are subcomponents of the MRL subsystem to modes of the Shuttle guidance system. Note that much of the information about the changes that occur in an MRL release process is not captured in this GOA (e.g., power consumption effects are not of interest).

## *Conclusions*

Functional models extend the power of behavioral models because they can express system goals as well as structure and behavior. Their properties make them especially useful for supporting the development and validation of fault management procedures. Three points deserve special emphasis.

First, types of conditions in the regulated process correspond to some parts of the qualitative process definition of Forbus. Individuals, preconditions, and quantity conditions of a Forbus process correspond to system components, connections, and state variables that are conditions in the regulated process. Influences and qualitative proportionality relations in a Forbus process will correspond to modulators in the regulated process, but this is an area for future research.

Second, the concept of a controller as a verifier of contended conditions and intended goals sheds new light on concepts of coordinated action among teams, gained from analysis of the heterogeneous human-machine team in a designed system. Here, the "suspicion" is not an adversarial one about whether goals are shared between the human and the device, as described by (Levesque, 1990). Instead we provide a general framework for representing monitoring of goal-oriented activities based on selected conditions and outcomes.

Finally, functional models can support goal-directed simulations with explicit mechanisms to react to changing events. Nilsson (Nilsson, 1989) calls this *teleoreactivity* and uses it to make "smart" processes whose preconditions include the negated goal (e.g., if a process has a goal *g*, then ~*g* must be true for the process to begin). Functional modeling extends this notion to conditions as well as effects and does so in a fault management environment.

*References*

Balkanski, Cecile T. (1990) *Modeling Act-Type Relations in Collaborative Activity*, TR-23-90, Harvard University Center for Research in Computing Technology.

Davis, Randall, and Walter Hamscher (1988). "Model-based Reasoning: Troubleshooting." *Exploring Artificial Intelligence: Survey Talks from the National Conferences on Artificial Intelligence* (H. Shrobe, ed.) Morgan Kaufman Publishers, San Mateo, California, pp. 297-346.

Forbus, Kenneth D. (1984). "Qualitative Process Theory." *Artificial Intelligence*, 24 (1984) pp. 85-168.

Kant, Elaine (1988). "Interactive Problem Solving Using Task Configuration and Control," *IEEE Expert*, Winter 1988, pp. 36-49.

Levesque, H. J., P. Kohen, and J. Nunes, (1990). "On Acting Together." *Proceedings of the Eighth National Conference on Artificial Intelligence*, pp. 94-99.

Malin, J. T., Bryan D. Basham, and Richard A. Harris (1990) "Use of Qualitative Models in Discrete Event Simulation for Analysis of Malfunctions in Continuous Processing Systems." *Artificial Intelligence in Process Engineering* (M. Mavrovouniotis, ed.), Academic Press, pp. 37-79.

Muratore, J., T. Heindl, T. Murphy, A. Rasmussem, and R. McFarland (1990). "Real-Time Data Acquisition at Mission Control." *Communications of the ACM*, December 1990, Volume 33, number 12, pp. 19-31.

Nilsson, N. J. (1989) *Teleo-Reactive Agents*. Computer Science Department, Stanford University (forthcoming).

# Autonomous Power System Intelligent Diagnosis and Control

Mark J. Ringer
Todd M. Quinn
Anthony Merolla
Sverdrup Technology Inc.
NASA Lewis Research Center Group
Brook Park, Ohio 44142

## Abstract

The Autonomous Power System (APS) project at NASA Lewis Research Center is designed to demonstrate the abilities of integrated intelligent diagnosis, control and scheduling techniques to space power distribution hardware. Knowledge-based software provides a robust method of control for highly complex space-based power systems that conventional methods do not allow. The project consists of three elements: the Autonomous Power Expert System (APEX) for fault diagnosis and control, the Autonomous Intelligent Power Scheduler (AIPS) to determine system configuration, and power hardware (Brassboard) to simulate a space based power system.

The Autonomous Power Expert (APEX) is a software system that emulates human expert reasoning processes to detect, isolate, and reconfigure in the case of a power system distribution fault. The APEX system continuously monitors the operating status of the Brassboard and reports any anomaly (either static or incipient) as a fault condition. APEX functions as a diagnostic advisor aiding the user in isolating the probable cause of the fault. Upon isolating the probable cause, APEX automatically reconfigures the Brassboard based upon internal knowledge as well as information from the scheduler. APEX provides a natural language justification of its reasoning processes and a multi-level graphical display to depict the status of the Brassboard.

The Autonomous Intelligent Power Scheduler (AIPS) is an intelligent scheduler used to control the efficient operation of the Brassboard. A database is kept of the power demand of each load on the Brassboard and its specified duration and priority. AIPS uses a set of heuristic rules in order to assign start and end times to each load based on priorities as well as temporal and resource constraints. When a fault condition occurs AIPS assists APEX in reconfiguring the system.

The APS Brassboard is a prototype of a space-based power distribution system and includes a set of smart switchgear, power supplies and loads. Faults can be introduced into the Brassboard and, in turn, be diagnosed and corrected by APEX and AIPS. The Brassboard also serves as a learning tool for continuously adding knowledge to the APEX knowledge base.

This paper describes the operation of the Autonomous Power System as a whole and characterizes the responsibilities of the three elements: APEX, AIPS and Brassboard. A discussion of the methodologies used in each element is provided. Future plans are discussed for the growth of the Autonomous Power System.

## Introduction

Our future presence in space will require larger and more sophisticated working and living environments. Such environments will consist of numerous integrated subsystems that will have to be maintained with a high degree of reliability. The electrical power system on a platform such as the Space Station Freedom, Lunar base, or Mars base will be one such subsystem. The APS project explores intelligent hardware and software architectures for safe and efficient system operation.

On large space platforms, the small number of humans on-board will be overloaded with science and other activities. Normal operational concerns of the power system including the control of a large number of switching devices, routine maintenance checks, and scheduling of loads would overwhelm the crew of any such mission.

Ground based experts are one possibility for control. The operators on the ground would be responsible for the minute-by-minute operation of the power system. These experts will be expensive, and the lifecycle cost of any such system would surely suffer in the long run, if too much work rested on the shoulders of these experts. Also, these experts would have a long problem solution time, if they are immediately available at all. This problem is compounded by communication delays, if the power system is more distant than low earth orbit [Dolce].

Since the control of such a large system is difficult to accomplish by the use of on-board crew or ground based systems, the importance of an automated on-board system is evident. The more responsibilities that an automated system assumes will increase the speed of critical decisions. Consequently, this will improve reliability and safety, and drastically lower the life-cycle costs of the power system. The expertise needed to automate these systems could be contained within a set of expert systems. For this reason, the use of expert systems for control of such a power system will significantly decrease operating costs, increase efficiency, and provide for safer operation.

The proposed distributed design of the power system will be much different than bus-based power systems of earlier spacecraft. The operation of a large distributed space power system will be more efficient and safer than its bus-based counterpart, although its control will be more complex.

Hardware design will be a major element in keeping the system operating as safely as possible. Safing of the power distribution system in the case of a "hard" or catastrophic fault will be accomplished by embedded controllers and "smart" switchgear. To preserve the efficient operation of the power distribution system, however, the fault must be isolated, and appropriate recovery procedures must be performed. This is the job of the fault diagnosis expert system. Potential power disruptions can also be avoided by detecting incipient fault conditions that are not immediately threatening to the power distribution system, but over a period of time will become a fault. Soft faults, which can cause graceful degradations of a system, are also best detected by expert systems.

Intelligent scheduling systems will be an integral part of minute-by-minute operations. The scheduler must know the state of the system at all times and be able to help reconfigure in the case of a fault. The most important responsibility of the scheduler is the efficient operation of the system which is accomplished by assigning as many of the available resources as possible to the proper activities.

The object of the APS project is to demonstrate the use of intelligent control, diagnosis and scheduling technologies to a prototype space power system. Many of the obstacles encountered in the development of software are attributed to the integration of multiple software products, and further integration of these products with hardware. These obstacles as well as aspects of communication, cooperation, and protocol will be addressed. This paper describes many of the potential design and development concerns of an automated space-based electrical power system.

## Brassboard

### Power System Design

In a distributed power system such as a terrestrial power utility, there are multiple levels of control, including power generation, long range high-voltage transmission, electrical substations, down to commercial and residential load centers. These sub-systems, collectively known as the power grid have controllers ranging from computers operating at regional control centers to humans turning on wall switches. The higher level sub-systems have built-in redundancy to keep the system operating in the event of a local fault. Power can be used at will, as spinning reserve generators are available to generate more power when needed. The complexity of control of such a system is proportional to the relatively small number of power generating facilities. The power utility, in most cases, does not have to worry about which users are demanding power from the grid, only a measure of loading on the grid as a whole.

In a conventional bus-based satellite power system, the operating scheme is much different. The power system is designed so that it has just enough power to run all the power consuming loads at the peak operating condition. All of this power is supplied to a local bus to which all loads are connected. There may be some redundancy in the form of multiple busses, but the architecture is usually quite simple. In general, there is no need to schedule loads, since there is sufficient power to meet load demand. This architecture does not provide for a robust and reliable control environment when designing a large power system.

The proposed distributed power system for space applications is much like that of a terrestrial power utility with regard to power generation, transmission, and distribution. Unlike a terrestrial system where power is supplied to a load on demand, all required power must be determined in advance by the scheduler. In the space system, the total amount of resources is fixed, and the loads must be controlled. Since the number of loads is much larger than the number of sources, the complexity of this control scheme is much higher. This complexity is offset by the added flexibility, reliability, and safety offered over a

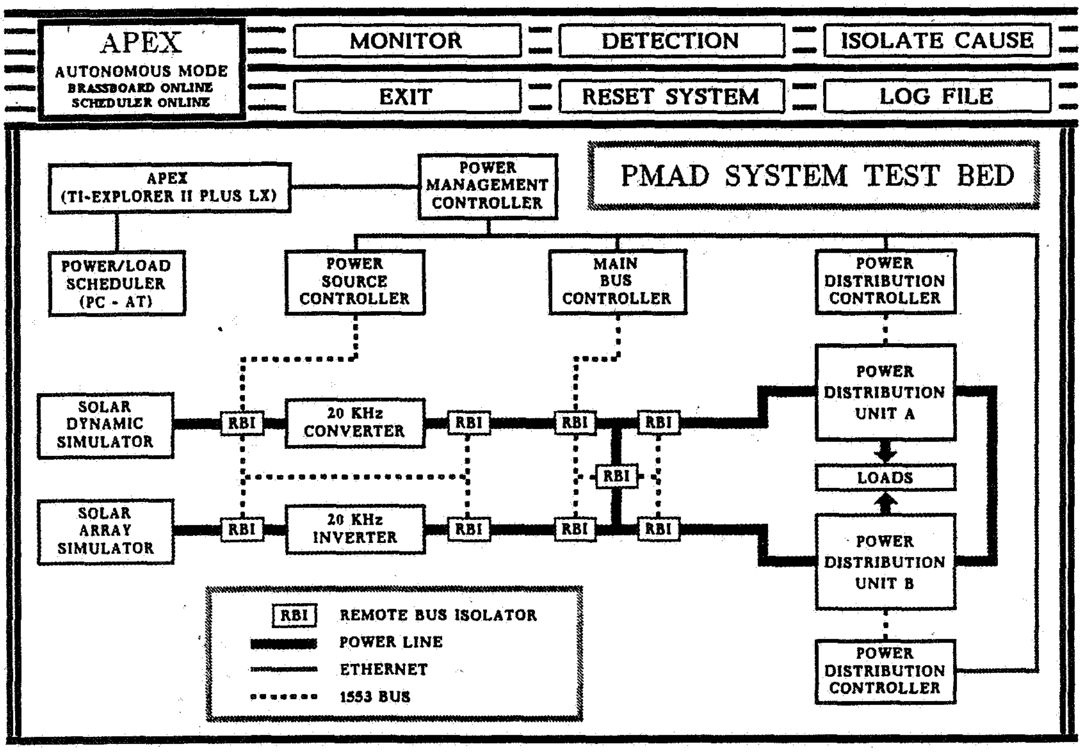


Figure 1. Power System Distribution Architecture

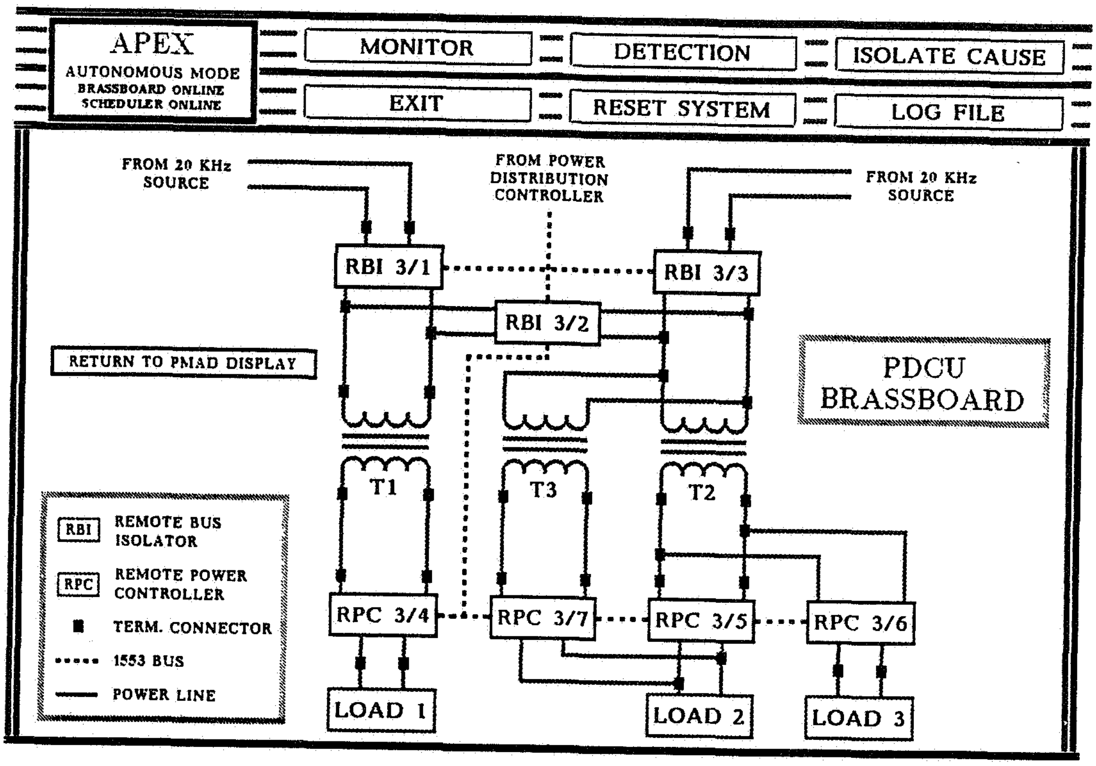


Figure 2. APS Brassboard Configuration.

conventional bus-based system.

The power system switchgear are designed to safe the system in the case of a hard fault as quickly as possible. This is done in a time frame much shorter than possible by any type of software. After the system is safed, a software system can then diagnose and reconfigure the system to return it to a safe and efficient operating condition. Another important design feature in a terrestrial utility as well as the proposed space-based design is the concept of coordination. Coordination prevents faults from propagating up the system. This means that if a fault occurs at some level of the distribution structure, it should be isolated at the next higher level. Without such a design, any fault could disable the entire system. Containment of these faults is accomplished by designing the lower level switches to trip faster and at lower current levels than the higher level switches. This safes the system and keeps a local fault from affecting a large portion of the power distribution structure.

The space power system described in this paper is based on such a distributed system architecture. This architecture is based on a previous Space Station Freedom design and the architecture is the important aspect of the system [Beach]. The operating frequency does not affect the function of the distributed control scheme, it will function with either an AC or DC system. The entire power system architecture is shown in Figure 1. The APS Brassboard is shown in Figure 2. Figure 3 shows a switch level diagram of an RBI.

## Hardware

Each piece of switchgear, contains a set of sensors, analog-to-digital electronics, 1553 bus interface, and a set of power supplies. Two types of switches are used on the Brassboard, Remote Power Controllers, RPCs, and Remote Bus Isolators, RBIs. An RPC contains only a solid state switch. An RBI contains both solid state and mechanical switches which allow for a higher power capacity and increased operating efficiency. The RPC switches on the order of 30 micro-seconds. In an RBI, the solid state switch turns on first, then the mechanical switch engages to lessen the effects of contact

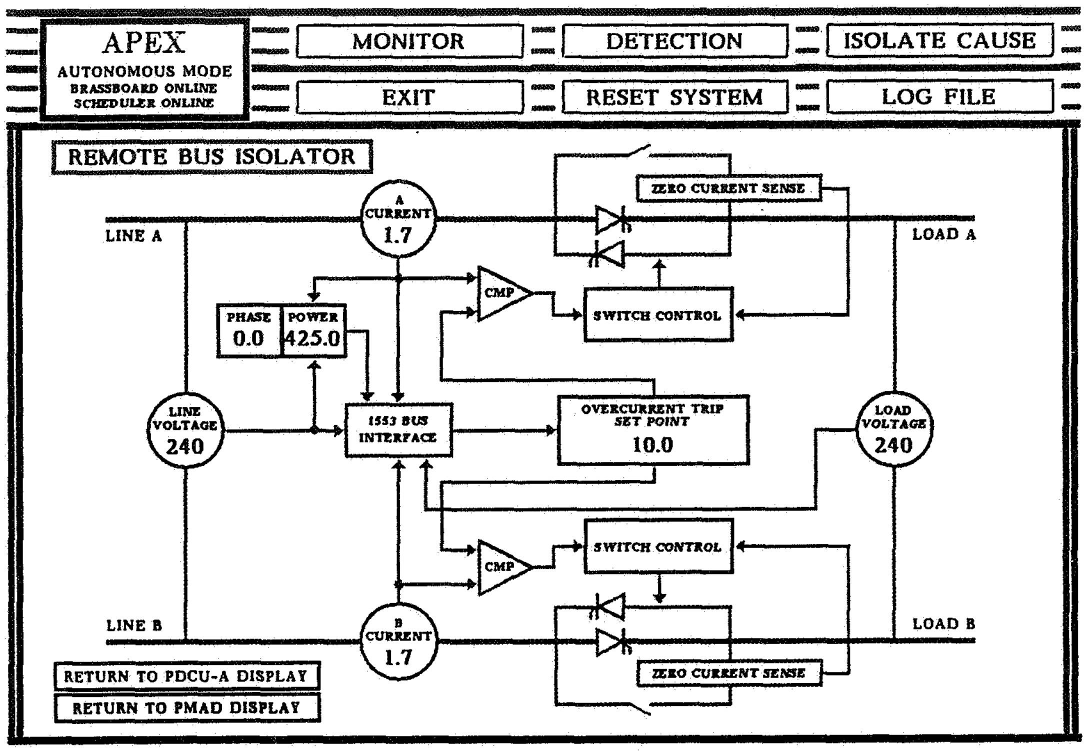


Figure 3. Switch Diagram of an RBI.

depletion and arcing, and to increase the efficiency at high power levels. This takes place within 15 milli-seconds. A set of 20 kHz transformers is also used to step down the voltage between the RBIs and RPCs.

The signal sensing and limiting circuit is the heart of the switches' "smarts". Voltage, current, power, and phase angle are monitored both at the line (power input) and load (power output) side of the switch. This information is then digitized using an 8-bit analog-to-digital signal converter. A circuit is included which stores the over-current set point which is determined by the controlling software. The circuit continuously monitors this limit, and if a problem is encountered, the switch is tripped. Since the switch has both line and load sensors, after a switch has tripped, data can still be taken to see if it is again safe to turn the switch on, or to find out exactly what the problem is.

The Brassboard has two 20 kHz power sources, a 1.5 kW power amplifier with an external oscillator and regulator circuit, and a 12 kW Mapham inverter. Loads on the brassboard are simple incandescent or infrared light bulbs. These loads provide a good visual indication of what switches are turned on. When current or voltage problems exist, a dimming of the lights is produced.

## Brassboard Communications and Software

The brassboard switchgear is controlled by a set of Intel 8086 based single board computers. Each computer has a built in 1553B, IEEE 802.3, and RS-232 communication capability, which allows them to communicate with the switchgear, each other, and outside computers respectively. The Power Management Controller PMC and Power Distribution Controller PDC represent a hierarchy of controllers responsible for the Brassboard's operation.

The control algorithms within the PMC and PDC are implemented using embedded Ada language software [Wright]. APEX sends a request for data to the PMC over the RS-232 link. The PMC contains continuously updates values of voltage, current, power, phase, as well as 12 bits of status information for each piece of switchgear.

## Faults

The Brassboard normally operates

with the no faults existing in the system. To develop fault detection, recovery, and reconfiguration schemes, actual hardware faults must be introduced into the system. The APEX fault diagnostic system is capable of detecting hard, soft, and incipient faults.

A hard fault will cause a switch to trip, thus generating an abrupt change in system configuration. This type of fault can be caused by a multitude of possible problems including switch failures, short circuits across power lines, and some types of load failures. These can be simulated by switches in the system to cross the power lines, etc.

A soft fault is a type of fault which does not cause a switching device to trip. It can be manifested as a small leakage path in the wiring between switches, problems within the load itself, leakage paths within the switch itself, or problems with parts of the mechanical switch. This can be simulated by introducing various resistive circuits into the brassboard.

Incipient faults are faults that can be detected before they actually become a problem. Such faults could be small changes in a load power demand, a small degradation in the insulation of wiring, or small resistive faults building up within the switchgear. The incipient faults can be simulated by the introduction of a programmable load to portions of the Brassboard.

### Reconfiguration

The Brassboard architecture, although simple by comparison, incorporates many of the design concepts found in a much larger system. The Brassboard includes two power sources to which any load can be attached. If one source needs to be disconnected from the system or fails, the remaining loads can be reconfigured and attached to an alternate power source. In this way, a level of redundancy exists for dealing with source faults. One of the loads on the Brassboard has a redundant path attached to it. This is meant to simulate a fault of one piece of switchgear, and loss of power to the load. This fault can be cleared by switching the load over to the redundant path.

## APEX

### Introduction to Operation

The APEX system is designed to take the place of a human expert in the control and diagnosis areas that require intricate thought patterns, repetitious operations, rapid response time, or where human experts are not available. The Autonomous Power Expert (APEX) system has been developed to emulate human expert reasoning processes used in the diagnoses of fault conditions in the domain of space power distribution. The knowledge that exists in the minds of the power system experts and their thought processes represent a sound method to diagnose faults in such a power system. In order to capture this method, isolation, diagnosis, and reconfiguration knowledge is represented in APEX as a set of rules and data concerning the operation and configuration of the power system. APEX is connected to the Brassboard and the scheduler in order to implement an entire power system control architecture.

APEX receives data from the PMC and then initiates fault detection. APEX detects faults by comparing expected values obtained from AIPS to the measured operating values obtained from the Brassboard. If no deviations from the expected operating state of the Brassboard are found, APEX will again request data from the PMC and re-initiate the fault detection activity with the new data.

Upon detection of a fault condition, APEX accesses information and rules contained in its knowledge base, reaches a conclusion, and displays the probable cause for the detected fault to the user. Actions are then taken to correct the fault and return the Brassboard to a safe and efficient operating condition. The user can also ask APEX to justify its conclusion. APEX can operate as either an advisor or an autonomous controller of the APS Brassboard.

## Implementation Overview

APEX is currently implemented on a Texas Instruments Explorer II workstation in LISP and employs the Knowledge Engineering Environment (KEE) expert system shell. APEX consists of a knowledge base, a database, an inference engine, and various support and interface software. The knowledge base comprises facts and rules that correspond to knowledge acquired from the human expert during problem solving. The database is the basic working area where storage and calculations of sensory data for incipient fault detection occurs. The inference engine is the reasoning mechanism that draws conclusions from information stored within the knowledge base. In choosing the appropriate recovery procedures for the isolated fault, APEX also relies on the reasoning capabilities of the inference engine. Conventional software provides the user with an interactive interface to communicate with APEX and to obtain data from the Brassboard and AIPS.

## User Interface and Operation

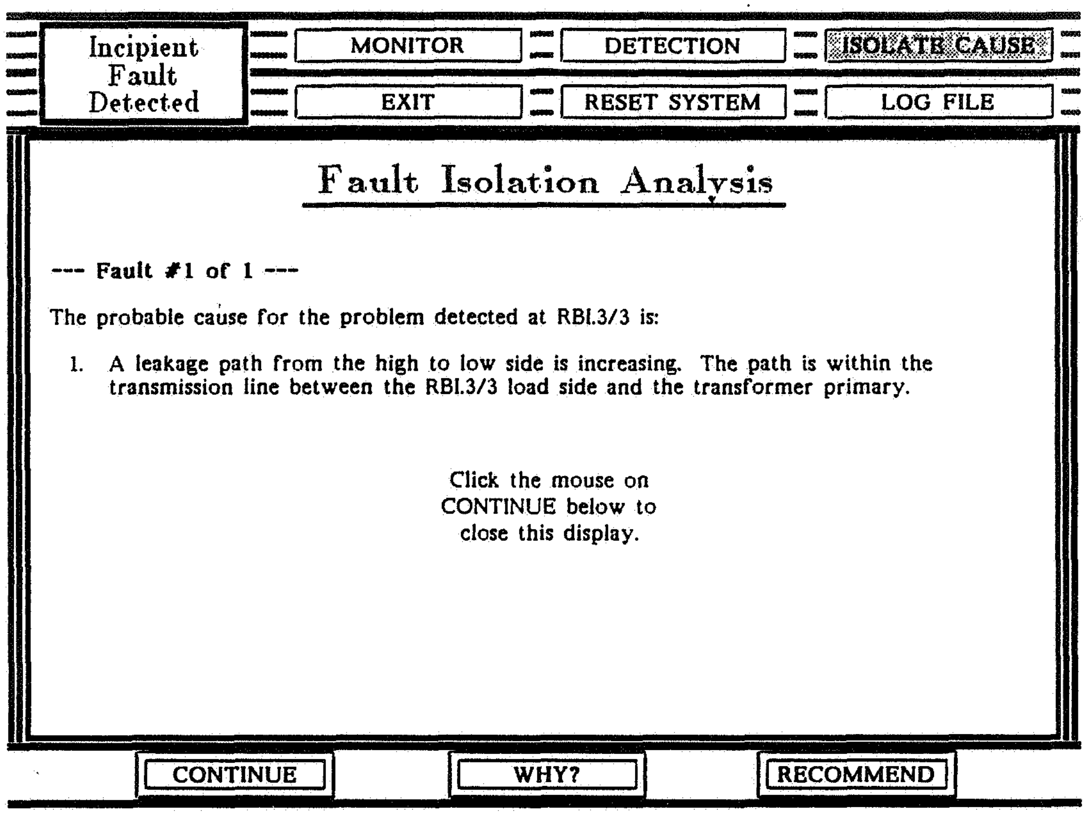
The goal of the user interface is to provide access to APEX which requires a minimal amount of training. Communication between APEX and the user is accomplished by easy-to-use mouse-selectable menus, color graphics and text displays. The user interface screen presents a color display that is divided into three areas as shown in Figure 1. The top portion of the screen is the control menu that allows the user to select the desired APEX function. When a function is selected, mouse-selectable options for that function appear in the options menu located in the lower portion of the screen. Located on the left side of the control menu is the APEX mode/interface menu. Fault detection and fault isolation results are shown within the main display area by means of color diagrams and text explanations.

The graphical displays in the main display area consist of a set of hierarchical diagrams that represent three different levels of the power distribution system. The diagram in the main display area shown in Figure 1 represents the overall power distribution system. When an active fault is detected, the area of detection is outlined in red. For an incipient fault condition, the area is outlined in yellow. The yellow indicates that a parametric value is probably going to go out of tolerance if preventive action is not taken. The user can get a more detailed diagram of an area by choosing the particular area of interest and clicking the mouse. Figure 2 shows the Brassboard top level diagram. Figure 3 shows the switch level diagram which is one level below the Brassboard diagram. Each switch level diagram displays the actual measured data values enabling the user to see which parametric attribute is out of tolerance.

The control menu in Figure 1 contains the following six mouse-selectable functions: *monitor, detection, isolate cause, reset system, log file*, and *exit*. The *monitor* selection causes APEX to continuously acquire and check parametric values from the power distribution system. When either an active or incipient fault is detected, APEX displays a "fault detected" message in the upper left corner of the user interface screen. Once alerted, the user can display the fault detection analysis by selecting *detection* in the control menu. When *isolate cause* is selected from the menu, APEX will access the fault isolation rules to determine the probable cause of the detected fault.

Recall that when a function is selected, the options menu provides the user with available options for that function. For example, when the user selects *isolate cause* from the control menu, APEX will display the probable cause of a detected fault and the options menu will contain *continue, why?, recommend*. This is shown in Figure 4. If the user selects *why?*, APEX will display the reasoning process leading to the probable cause conclusion as shown in Figure 5. The *recommend* option allows the user to request recommended action procedures for correcting the fault.

The mode/interface menu provides controls for selecting the operational mode of APEX as well as changing the online/offline status of the data acquisition and scheduler interfaces. APEX can operate in manual



Fault Isolation Analysis

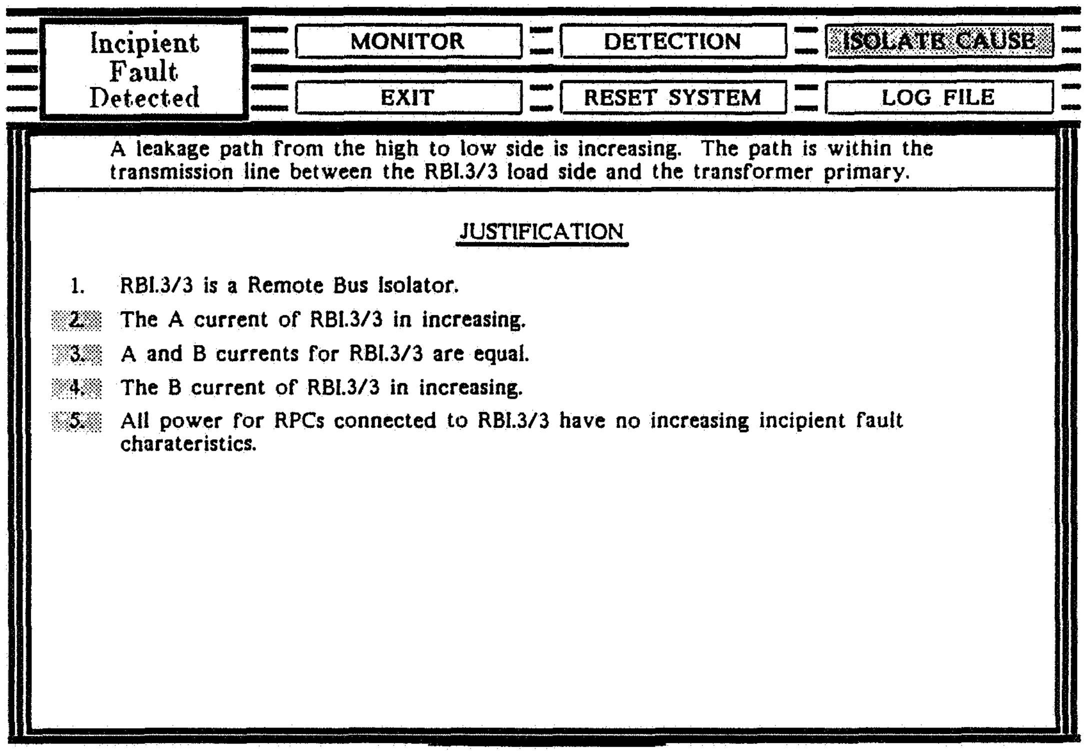
--- Fault #1 of 1 ---

The probable cause for the problem detected at RBI.3/3 is:

1. A leakage path from the high to low side is increasing. The path is within the transmission line between the RBI.3/3 load side and the transformer primary.

Click the mouse on
CONTINUE below to
close this display.

**Figure 4. Isolation.**



A leakage path from the high to low side is increasing. The path is within the transmission line between the RBI.3/3 load side and the transformer primary.

JUSTIFICATION

1. RBI.3/3 is a Remote Bus Isolator.
2. The A current of RBI.3/3 in increasing.
3. A and B currents for RBI.3/3 are equal.
4. The B current of RBI.3/3 in increasing.
5. All power for RPCs connected to RBI.3/3 have no increasing incipient fault charateristics.

**Figure 5. Justification.**

mode where the user selects appropriate commands from the control menu. In autonomous mode, APEX will monitor the power distribution system, detect faults, isolate the probable cause and provide appropriate fault recovery automatically without input from the user. The user can select whether APEX is to acquire data from the Brassboard or the Brassboard simulator. In scheduler online mode, APEX can request and receive live scheduling information from AIPS. When the scheduler interface is offline, APEX reads pre-saved scheduling information.

Brassboard Simulation, Display and Control

Part of the user interface, is the Brassboard data simulator/display/control interface. In simulation mode the main display area contains a diagram of the simulated Brassboard which can be set by the user. Along with simulated switching device data, the simulated data for each load on the brassboard is also displayed. With the use of the Brassboard simulator, the fault diagnosis capabilities of APEX can be used without the Brassboard. In display mode, actual sensor data from the brassboard can be displayed and recorded for the user's observation as shown in Figure 6. This function is used as an interface for our experts and operators to verify the reasoning of APEX. In control mode the user has the capability of issuing commands to the Brassboard in order to turn switching devices on/off and set trip limits.

Rule Base Operation

Representation of knowledge within the APEX knowledge base consists mainly of frames, semantic triples and production rules. Frames are structures which describe objects or classes of objects and the relationships between them. Objects are composed of slots which specify the different attributes belonging to each object. Individual slots of an object can contain either declarative or procedural information. Declarative information expresses facts about the object, whereas procedural information is in the form of a program or a set of procedural steps. Frames themselves are considered to be declarative information. Within APEX, declarative information is also represented by semantic triples which state information in the form of object/attribute/value (i.e. attribute of object = value). Production rules are 'If-Then' statements which infer either declarative or procedural facts when the conditions contained in the premises of the rule are found to be true. Again the facts are represented as a semantic triple (declarative) or a program (procedural).

The inference engine is the heart of the expert system, determining how knowledge is represented and processed. By operating on rules within the knowledge base, the inference engine can reason and make inferences about the state of the power system. The ways the inference engine processes the rules and data are commonly referred to as forward and backward chaining. Forward chaining (also known as data driven) works from the given data to a conclusion. Backward chaining (also known as goal driven) works from a particular goal and tries to either confirm or refute its truth. In the APEX system, fault detection is implemented with forward chaining and fault isolation is accomplished with backward chaining [Ringer].

Fault Detection

When a fault occurs in the power distribution system it appears as a deviation on the expected values of the currents flowing through the switching devices. Therefore, faults are detected by comparing the measured operating current values to the expected current values calculated by APEX and AIPS. A fault is detected when an operational value and the corresponding expected value are in disagreement within a given tolerance. With this mode of fault detection, the set of detection rules can be kept small, greatly reducing the time for fault monitoring.

Fault Isolation

The primary function of fault isolation is probable cause determination for a given fault condition. APEX uses the knowledge contained within the fault isolation rules and the backward chaining capabilities of the KEE inference engine to determine the most probable cause. Figure 4 shows a display of fault isolation analysis. In this case, there are three possibilities listed as the probable cause. Based upon the present knowledge in the knowledge base and the sensor data obtained from the power distribution system, the exact cause cannot be further isolated without more information.

Figure 5 shows a typical display of probable cause justification. At the top of the main display area the probable cause, which is the backward chaining goal, is displayed. Below the stated probable cause are the premises which support the truth of the probable cause statement. The unhighlighted numbers (1 - 4) are primitive statements of fact contained within the knowledge base. Numbers that are highlighted represent statements of facts that were inferred as subgoals. The user can see the premises used to prove the truth of each subgoal.

### Fault Recovery

After the probable cause of a detected fault or incipient fault condition has been isolated, APEX will analyze available information about the current operating conditions and take appropriate actions. These actions pertain to both short and long-term recovery. Short-term recovery determines if the power distribution system can be reconfigured, or if load shedding is necessary. For long term recovery, the repair procedures needed to correct the fault are determined after short term actions have been implemented.

Short-term recovery analysis is based on a set of "recommended action" rules for the particular fault condition. Information about available power sources, current configuration of the power distribution system, the scheduled run times of the loads, and the effects of the fault on the system are all considered during the analysis. If the fault is seriously affecting the amount of power reaching a particular load and an alternate path for power distribution exists, then the system can be reconfigured automatically, to allow the load to run to completion. When the fault cannot be tolerated and alternate power distribution paths are unavailable, the schedule for the loads is revised by the scheduler possibly resulting in load shedding.

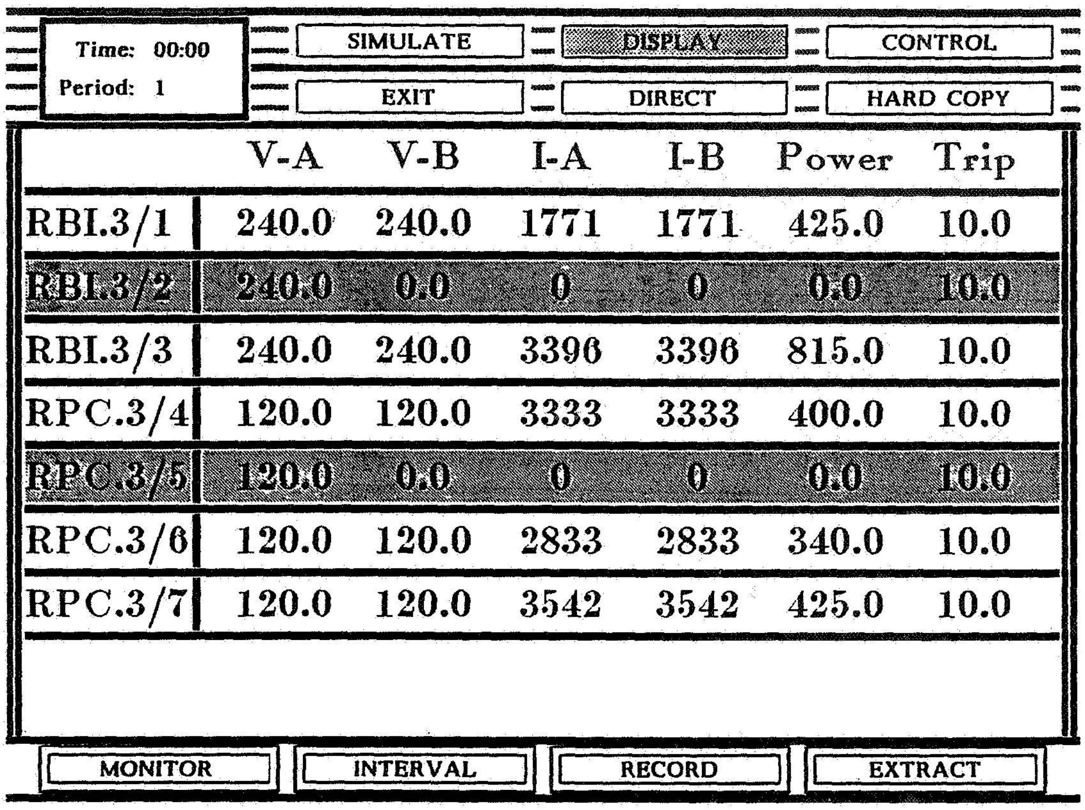
After short-term recovery, the cause of the fault in the power distribution system needs to be repaired. The appropriate procedures needed to repair the problem are determined by long term recovery, based on a set of recommended action rules. In some cases, the cause of the fault is traceable to a group of possibilities, and additional troubleshooting procedures are displayed to intelligently guide the user to further isolate the exact location and to make repairs.

### Incipient Fault Detection

If, during conventional fault detection, the rules have been exhausted with no faults detected, APEX checks for incipient faults. Incipient detection is based on statistical linear regression and correlation analysis of the historical data. Measured and expected parametric values of the power system are saved in a historical database. The expert system analyzes the historical data looking for any indication of a parametric attribute that has maintained either an upward or downward trend in the data values over a period of time.

Since the power system is dynamic and the measured value fluctuates over a period of time during normal operation, a parametric ratio of the measured-to-expected value is used to identify any increasing or a decreasing trends in the parametric data. Once the data has been stored in the database, correlation coefficients are calculated for each parametric attribute of all switching devices. A high correlation coefficient indicates that an incipient fault exists in the system.

Once an incipient fault condition has been detected, the user can view the results of the statistical analysis. The results are shown graphically to the user showing trends in the ratio between measured values and expected values. The output screen also shows in which switching device the trend is located. Along with the plot of the linear regression results, the



Time: 00:00
Period: 1

| | V-A | V-B | I-A | I-B | Power | Trip |
|---|---|---|---|---|---|---|
| RBI.3/1 | 240.0 | 240.0 | 1771 | 1771 | 425.0 | 10.0 |
| RBI.3/2 | 240.0 | 0.0 | 0 | 0 | 0.0 | 10.0 |
| RBI.3/3 | 240.0 | 240.0 | 3396 | 3396 | 815.0 | 10.0 |
| RPC.3/4 | 120.0 | 120.0 | 3333 | 3333 | 400.0 | 10.0 |
| RPC.3/5 | 120.0 | 0.0 | 0 | 0 | 0.0 | 10.0 |
| RPC.3/6 | 120.0 | 120.0 | 2833 | 2833 | 340.0 | 10.0 |
| RPC.3/7 | 120.0 | 120.0 | 3542 | 3542 | 425.0 | 10.0 |

Figure 6. Display Screen

correlation coefficient, slope, standard error, and y-intercept are displayed for the user. The next step would be for APEX to initiate the fault recovery phase.

### Scheduler Interface

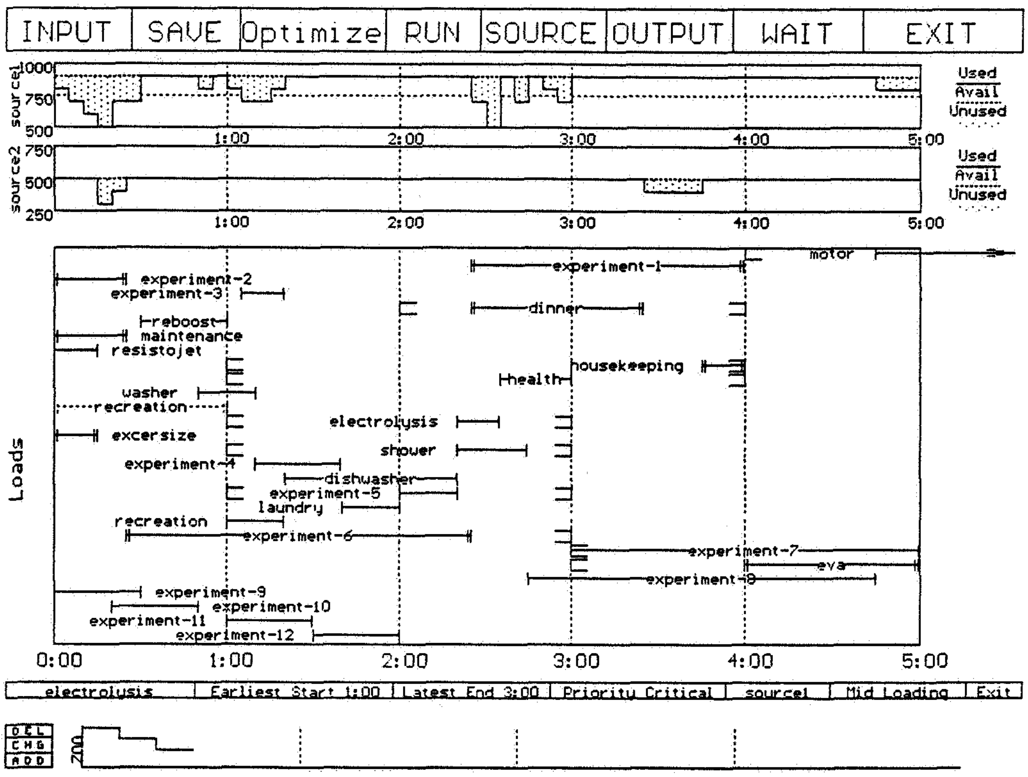
The scheduler interface is responsible for data exchange between APEX and AIPS. APEX initiates a request for a load schedule by transmitting the profile data to the scheduler over an ethernet connection. AIPS determines a schedule of starting times for each load and returns the information back to APEX. APEX uses the received schedule along with the load profiles as the basis for its expected value calculations.

## Scheduler

### Introduction

The distribution of resources aboard a space based system differs greatly from what is typical on Earth. An increase in power generation to match the overall load is common in terrestrial power systems. In a space-based system, all resources are limited. Therefore, the number of loads need to be adjusted to fit the available amount of resources. The limited supply of resources forces each requester to document which resources are needed and in what quantity. The scheduler takes in all of these requests and information on available resources, and generates a timeline of what activities can occur at what times without violating any constraints.

In a continuously manned system, there may be a few dozen resources available, and a few hundred loads that must be scheduled each day. A representation of this as a mathematical optimization problem is possible, but the solution is NP Hard. An NP Hard problem is one in which a linear increase in the problem size increases the complexity of the solution at an exponential rate. A mathematical optimization solution could take centuries of computer time even on a fast machine. Since the system will be operational around the clock, this scheduling must be done in real-time. This real-time is defined by the fact that a schedule for a days



electrolysis | Earliest Start 1:00 | Latest End 3:00 | Priority Critical | source1 | Mid Loading | Exit

Figure 7. Scheduler Output.

activities must be developed in a day or less. If a schedule takes longer than this definition of real-time, the scheduler will always be backlogged. For this reason, other methods of schedule generation must be employed, that will then give a solution to the problem within our definition of real-time.

Many scheduling methods, referred to as engines, capable of producing acceptable solutions in a reasonable amount of time exist. Constraint propagation, simulated annealing, and various types of heuristic based scheduling approaches all produce good results. AIPS uses a set of heuristics to construct a schedule. This method takes a set of loads and, based on a set of rules, places each of these loads on the schedule. This schedule can be generated very quickly with one depth-first path through the search space.

This heuristic engine can produce a good schedule very quickly, in fact, sometimes even faster than needed. In order to take advantage of this extra time, the one-pass engine can be augmented by an optimization engine. This optimization engine can be employed until the time that the schedule must be implemented. It is also important that the scheduler be able to stop at anytime during the optimization process and produce a feasible schedule. This is considered an anytime engine. A scheduling engine is considered "anytime" if at anytime during the solution, it can be stopped and a feasible schedule exists. This concept is important if the scheduler needs to operate in conjunction with other software and hardware that need the information as quickly as possible. The AIPS scheduler is always considered "anytime" since the scheduling engine does not allow constraint violations to exist at any point during the scheduling process [Zweben].

## Implementation

The Autonomous Intelligent Power Scheduler (AIPS) is designed to schedule a subset of the conditions that will exist on a space-based system. This will demonstrate the feasibility of integrating intelligent software systems for the control of a distributed space power system. AIPS determines the Brassboard configuration and assists APEX in reconfiguration when faults are detected. AIPS is completely autonomous needing no user input to completely schedule any set of activities received from APEX or when interactively operated. AIPS is written in the C language on a PC platform and is connected to APEX via an ethernet link.

In order to adequately model the interaction between APEX and AIPS, a set of protocols was developed to communicate different scheduling and rescheduling procedures. Protocols were developed to generate an initial schedule and modify existing schedules. The initial schedule generation takes a set of sources and loads and generates a schedule for APEX to follow. Five modification protocols exist: activity change, resource change, activity add, activity delete, and resource delete. During the execution of a schedule, the priority of an activity may change, the power demand of a load may change, the load may need to be dropped from the schedule, or a load may need to be added. Resources also may be changed during the execution of a schedule, or deleted altogether. These protocols enable APEX and AIPS to communicate system configuration information as well as reconfigure the system in the case of a fault.

### Internal Structure

Since the problem of scheduling the vast array of loads is so great, one easy way of reducing the complexity of the problem is to break the problem up into smaller sub-problems. In fact this is the only way to solve such a problem. Even if all of the loads that needed to be scheduled were known ahead of time, which they aren't, it would be an impossible task. The problem is currently broken up into a set of planning horizons. This necessitates the carry-over of information from one planning horizon to the next because some loads will carry over from the previous horizon.

Many attributes of constraints on activities exist in a scheduling environment. Some of these attributes have been used in the development of the AIPS scheduler [Britt]. These attributes are stored as structured data objects for each load. A time varying profile of power requested by the load is stored as well as the length. The earliest start time and latest end time can be specified for loads that wish to start and finish in a certain time window. The scheduling engine also take into account the priority of a load. Priorities are critical, normal, and low. A set of functions also exists to allow for loads to continue in from the last planning horizon, this is done by using the priority immediate. Each load is attached to a certain power source. It is not feasible that the activity can be assigned to more than one source given the architecture of the system. The last attribute specified for each load is a loading preference, declaring whether the would rather be scheduled as early as possible, as late as possible, or at the middle of its available start time window.

Evaluating the "goodness" of a schedule can be an ambiguous task. It is important to both schedule the high priority loads, as well as use as much available power as possible. What if it is possible to delete a higher priority load and replace it with a lower priority load which makes for an overall higher power use? Is this a better schedule? A balance between these and many other rating schemes must be decided. The AIPS scheduler takes into account many of these elements but, coming from a power system perspective, the AIPS scheduler gives preference to a schedule that uses more power than one that schedules more loads.

### Scheduling Engine

The main scheduling engine of AIPS is a heuristic one-pass engine. The information used to position each activity includes projected resource demand and total resource use, as well as each activities power demand, temporal constraints, requested loading preferences, and priority. Using this method, a schedule is generated in one-pass using information of how a human would build a "good" schedule. This may not be the mathematical optimum but this schedule can be generated in a very short time.

Since this is a one-pass engine it must be decided in which order to place the loads onto the schedule. This decision influences the final outcome of the schedule. In a one-pass engine, an activity has a much better choice of available positions, as well as a better chance of being scheduled if it is put on the schedule before most other loads have been put on. It is also advantageous to place larger higher power consuming loads on the schedule as early (in the scheduling process) as possible, because they may not fit if they are put on later, when much of the available power has already been subscribed. Also, if a load is very constrained in the length of its possible start times, it should be placed on the schedule earlier. This ranking is done before any load are actually placed on the schedule.

Since the amount of power demanded differs over time (temporal constraints of loads causes this), a projection of the loading on each power source at each time period can be made. This projection can then point out projected bottleneck areas for each source. The important point is that this is a projected demand, no loads have actually been scheduled yet [Biefeld]. Next, each load is placed on the schedule.

Each feasible start time where a load can be scheduled is determined and evaluated based on the previously stated factors. The projected demand here is used to influence the decision of where to place the load. Other factors are also used in the temporal placement of the load. A front loaded schedule will usually make for more resource usage (because it tends to pack the schedule more tightly), but this must be balanced with the conflicts of projected bottleneck regions, as well as each load's preference for a certain position on the timeline (front, mid, end loading). Based on these criteria, a start time is decided upon. This process is repeated until all loads have been attempted to be placed on the schedule.

Explanation of Output

The scheduler also has an interactive graphical user interface that can be used to test the scheduler as shown in Figure 7. This interface is fully mouse controlled and allows the user to edit activity information, resource information, as well as making changes to the schedule after the scheduling engine has given its solution. The user may test to see if they can manipulate the schedule in such a way to build a "better" schedule than the scheduling engine itself. This user interface also makes it easy to see how the scheduling engine works and makes it easier to test the scheduler.

The upper two graphs show the two sources of electrical power, with the power on the y-axis and time on the x-axis. Available power is shown as a dotted line, while scheduled power is shown as a solid line. This difference between available power and scheduled power is the unused (unscheduled) power which is shown as the dotted fill area between the available and scheduled power. This display can also show the amount of power that is oversubscribed. The user can introduce oversubscriptions in the schedule but this makes the schedule invalid.

The gantt chart in the middle of the screen shows each load that was scheduled. The length of the load corresponds to the length of the load and the scale is shown on the x-axis of the gantt chart. The earliest start and latest end points (if they are specified) are shown by brackets at each side of the load. On the color screen, each load is color coded by its priority. If a load continues into the next planning horizon, this is shown as an arrow at the end of the load.

The edit window at the bottom of the screen shows a more specific description of the load that the mouse is currently pointing to. Values such as power demand, length, start and end constraints, priority, source, and loading preference can be specified within this window.

## Future Developments

Rules need to be added to enhance the performance and increase the scope of APEX's knowledge in the areas of long-term recommended actions, autonomous mode operation, and recovery scenarios. Enhancements are needed in the areas of updating the log file, updating the user interface, and allowing time variant load operations. Additional software implementation for testbed control and schedule display are also needed. The application of model-based reasoning to the system is also being studied to relieve the overhead of a large number of rules and provide deeper reasoning for fault isolation.

The Brassboard currently contains seven pieces of switchgear, and two controllers. Future enhancements will greatly increase the number of switchgear and controllers available. This will allow more complex control and fault scenarios to be tested. Time varying loads will also be added to the Brassboard load structure to increase the complexity and provide APEX and AIPS with a more realistic load set.

The scheduler optimization engine needs to be upgraded, since the current engine is very simple. This will allow for a better schedule to be generated. Also, a more robust interface between APEX and the scheduler is needed.

## Summary

With the advent of larger and more complex space environments, more reliable and efficient subsystems must be designed. The APS project represents a power system automation scheme implemented through the use of intelligent software systems. Many of the problems faced in the design of a complete system were encountered in the APS project. The integration of a power system, fault diagnosis software, and scheduling software proved the feasibility of implementing such a control scheme. Of course, more complex hardware and software scenarios still need to be tested to further explore the workings of such a system.

## Acknowledgements

The authors would like to thank the Space Station Electrical Systems Division at the NASA Lewis Research Center for their hardware and software support. We would also like to thank Gene Lieberman for his work on the Ada software, and Walt Krawczonek for being our power system "expert". And, Jim Kish for his work in project management and coordination.

## References

[Beach] R.F. Beach, G.L. Kimnach, T.A. Jett, and L.M. Trash, "Evaluation of Power Control Concepts Using the PMAD Systems Testbed", In Proceedings 24th IECEC, 1989.

[Biefeld] E. Biefeld, L. Cooper, "Operation Mission Planner: Final Report", JPL Publication 90-16, March 15, 1990.

[Britt] D.L. Britt, A.L. Geoffroy, J.R. Gohring, "Managing Temporal Relations", Proceedings of the Goddard Conference on Space Applications of Artificial Intelligence, 1990, NASA CP 3068.

[Dolce] J.L. Dolce, J.A. Kish, and P.A. Mellor, "Automated Electric Power Management and Control for Space Station Freedom", In Proceedings 25th IECEC, AIChE 1990.

[Ringer] M.J. Ringer and T.M. Quinn, "Autonomous Power Expert System", In Proceedings 25th Intersociety Energy Conversion Engineering Conference IECEC, AIChE 1990.

[Wright] T. Wright, M. Mackin, and D. Gantose, "Development of Ada Language Control Software for the NASA PMAD Testbed", In Proceedings 24th IECEC, 1989.

[Zweben] M. Zweben, M. Deale, and M. Eskey, "Anytime Rescheduling", NASA Ames Artificial Intelligence Research Branch Technical Report.

# A MACHINE INDEPENDENT EXPERT SYSTEM FOR DIAGNOSING ENVIRONMENTALLY INDUCED SPACECRAFT ANOMALIES

Mark J. Rolincik
Goddard Space Flight Center
Greenbelt, MD
(301) 286-6754

## Abstract

A new rule-based, machine independent analytical tool for diagnosing spacecraft anomalies, the EnviroNET expert system, has been developed. Expert systems provide an effective method for storing knowledge, allow computers to sift through large amounts of data pinpointing significant parts, and most importantly, use heuristics in addition to algorithms which allow approximate reasoning and inference, and the ability to attack problems not rigidly defined.

The EnviroNET expert system knowledge base currently contains over two-hundred (200) rules, and links to databases which include past environmental data, satellite data and previous known anomalies. The environmental causes considered are bulk charging, single event upsets (SEU), surface charging, and total radiation dose.

## Introduction

In order to analyze spacecraft environmental anomalies effectively, a tremendous amount of information, databases and expert knowledge must be considered. Information regarding satellite design, specifications and orbital history need to be assimilated with previous anomalies data and environmental conditions, while addressing the specific circumstances of individual users.

Seldom are the environmental problems encountered by scientists rigidly defined, and thus they lack clear mathematical solutions. Under such circumstances, algorithmic programs are too limited by their sequential logic, becoming too cumbersome when trying to consider a wide range of variables of varying degrees of certainty. EnviroNET's Expert System is being developed as a new Artificial Intelligence (AI) technique to cope with this voluminous data and fluidity associated with spacecraft/environmental causality models.

Unlike it algorithmic predecessors, an expert system can be flexible in the way that it attacks complex problems. By virtue of its three basic parts (a knowledge base, a fact base, and a driver interface) an expert system more closely simulates the methods of human experts who use a combination of known, empirically derived formulae, hunches based on degrees of certainty and experience, and even judicious "fudging" when specific data is lacking. Figure 1 shows the expert system configuration.

The modularity of the expert system allows for easy knowledge and database updates and modifications. It not only provides scientists with needed risk analysis and probability diagnosing not found in the usual programs, but it is also an effective learning tool on environments, and the window implementation makes it very easy to use.

## THE KNOWLEDGE BASE

The knowledge base, with its set of rules, is what makes a rule-based expert system unique. Best thought of as an independent collection of "if...then" statements, the rules are created by experts in their respective fields and reflect the current level of human experience, along with its uncertainties. Under the weight of these rules, and by the use of multifield variables, an expert system can be said to "ponder the possibilities" presented by the databases and current knowledge which are too extensive to be readily assimilated by any single person. Rather than being limited to conclusions that must satisfy a set of tightly ordered mathematical statements, the system is free to offer suggestions, considerations, and likelihoods.

### Expert Rules

The rules in the EnviroNET expert system comprise the knowledge base used to diagnose spacecraft anomalies. The rules themselves were received from Harry C. Koons and Dr. David J. Gorney who are also developing an expert system for Aerospace Corporation, though using a hardware dependent system and a Texas Instrument commercial package.

The rule format used in the expert system is shown in figure 2. Each rule has a subject associated with it (in this case one of the four causes considered), a description of the rule, and then the actual rule itself. The rules also have what is termed a 'confidence factor' associated with the right hand side of each rule. Algorithms, which normal programs are limited to using, have a 100% certainty to them, and are a subset of the general heuristic rules which the expert system uses.

This aspect of the rule-based expert system is very important in diagnosing anomalous behavior since much of the knowledge, rules and experience required to diagnose these anomalies have probabilities associated with them. The use of such probabilities in the expert system introduces the concept of 'risk assessment' to the diagnostic procedure and

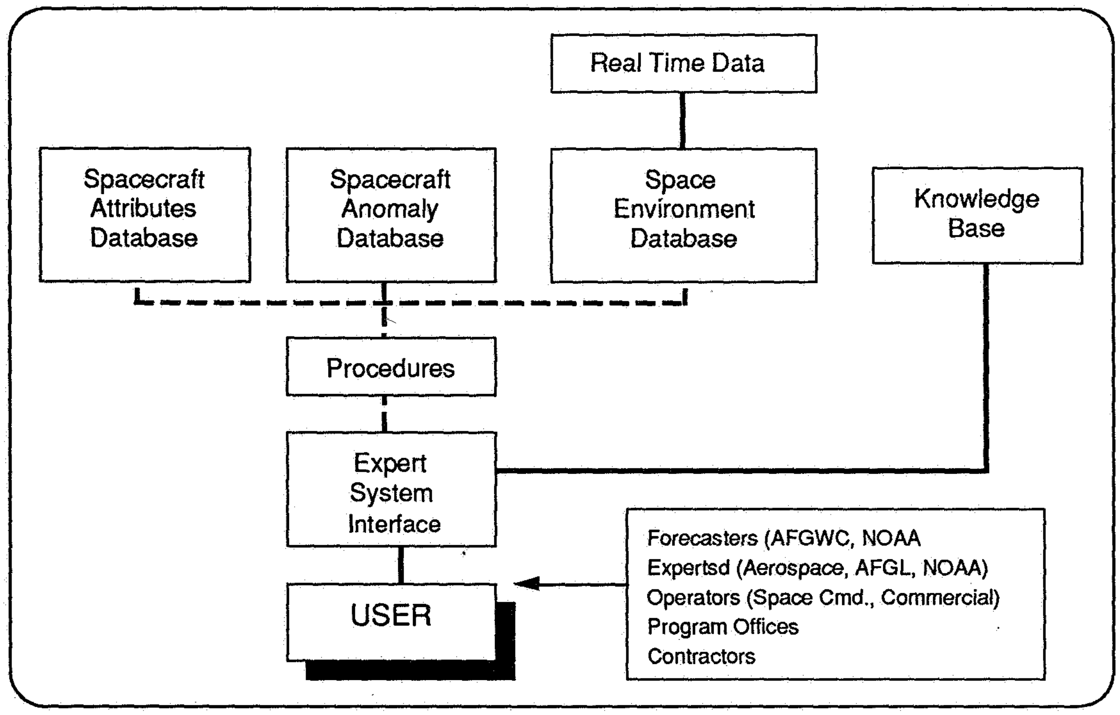

Real Time Data
Spacecraft Attributes Database
Spacecraft Anomaly Database
Space Environment Database
Knowledge Base
Procedures
Expert System Interface
USER
Forecasters (AFGWC, NOAA
Expertsd (Aerospace, AFGL, NOAA)
Operators (Space Cmd., Commercial)
Program Offices
Contractors

*Figure 1. Expert System Configuration*

the inclusion of knowledge which otherwise would be lost, since it is, at the very least, extremely difficult to represent such knowledge using mathematical formulae.

Another advantage of using a rule-based system is that it allows direct access to and easy comprehension of the knowledge and expertise used to diagnose the anomalies as opposed to the very complicated, and sometimes esoteric coding of most normal programs. Not only does this provide a way of storing the knowledge, but it also allows the system to be easily and quickly updated. These updates are accomplished by simply adding, deleting or modifying rules to which the system then automatically adjusts.

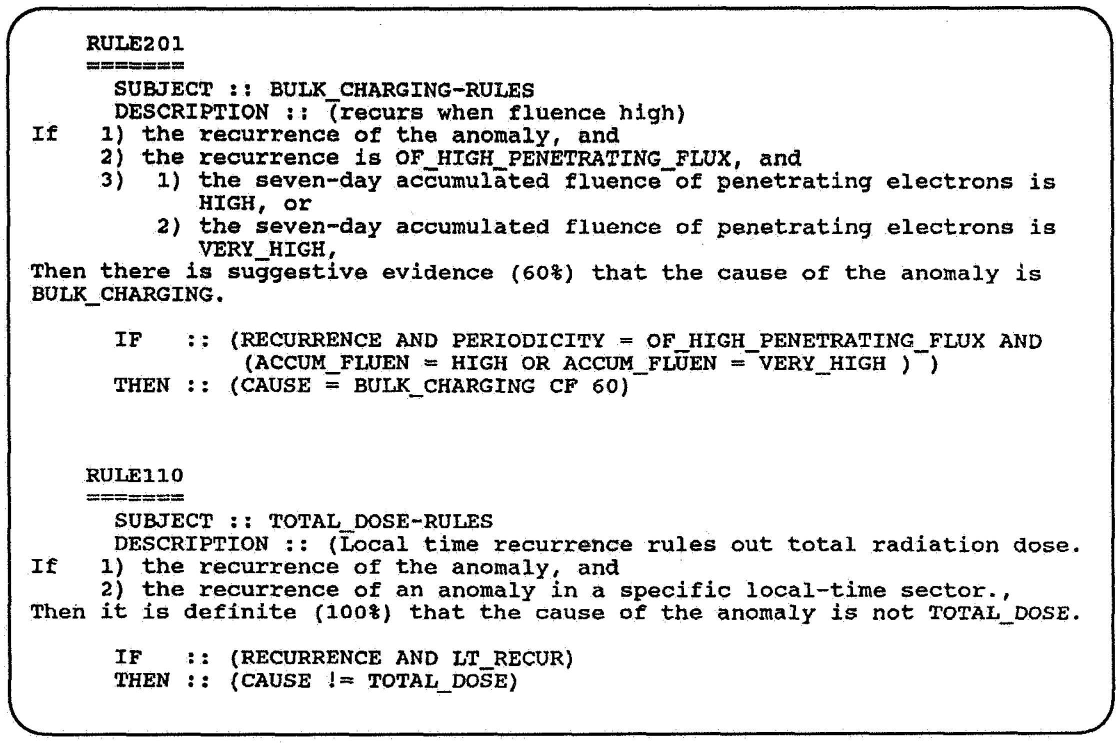
## Expert Shell

In order to facilitate the use of such rules, an environment must exist where they can reside and be accessed. Such an environment is called an expert shell. The expert shell that is being used by our system is called C Language Integrated Production System (CLIPS). CLIPS was developed by the NASA Johnson AI Laboratory in Houston, Texas. We chose CLIPS for many reasons, one of which is that the software is provided free of charge, unlike the expensive Texas Instrument's commercial expert shell that Koons and Gorney use.

Though CLIPS is considered a basic expert shell, its capabilities are more than adequate for handling the requirements of this system. CLIPS is not only compatible with both C and Fortran languages, but is machine independent, allowing it to be ported to EnviroNET's multiuser system. It also has features which include the ability to compile the rules and save them in a binary image file, thus allowing faster execution than a typical rule interpretive system.

## Variables

The EnviroNET expert system's use of variables is another area which makes this system unique, allowing it to handle non-



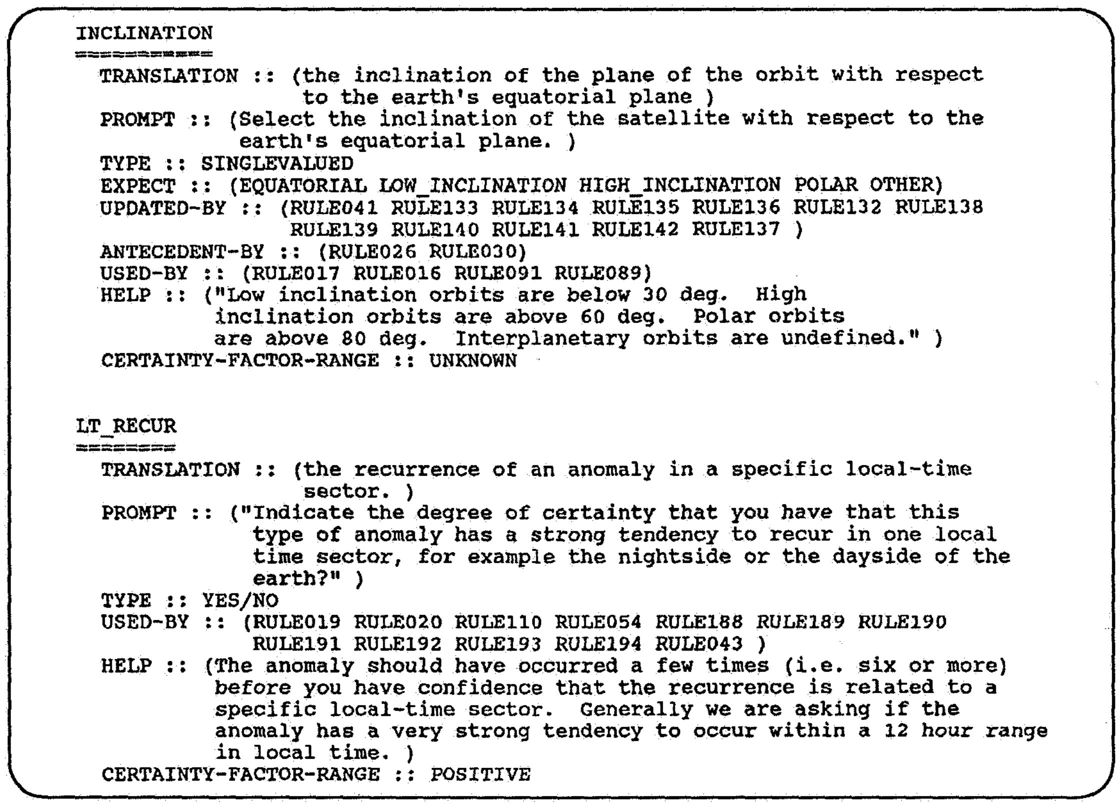
```
      RULE201
      =======
         SUBJECT :: BULK_CHARGING-RULES
         DESCRIPTION :: (recurs when fluence high)
If     1) the recurrence of the anomaly, and
       2) the recurrence is OF_HIGH_PENETRATING_FLUX, and
       3)   1) the seven-day accumulated fluence of penetrating electrons is
               HIGH, or
            2) the seven-day accumulated fluence of penetrating electrons is
               VERY_HIGH,
Then there is suggestive evidence (60%) that the cause of the anomaly is
BULK_CHARGING.

         IF   :: (RECURRENCE AND PERIODICITY = OF_HIGH_PENETRATING_FLUX AND
                   (ACCUM_FLUEN = HIGH OR ACCUM_FLUEN = VERY_HIGH ) )
         THEN :: (CAUSE = BULK_CHARGING CF 60)



      RULE110
      =======
         SUBJECT :: TOTAL_DOSE-RULES
         DESCRIPTION :: (Local time recurrence rules out total radiation dose.
If     1) the recurrence of the anomaly, and
       2) the recurrence of an anomaly in a specific local-time sector.,
Then it is definite (100%) that the cause of the anomaly is not TOTAL_DOSE.

         IF   :: (RECURRENCE AND LT_RECUR)
         THEN :: (CAUSE != TOTAL_DOSE)
```

*Figure 2. Rule Format*

algorithmic, equivocal problems. A variable in this system can take on one of three settings. It can be '*unset*', meaning that it has not been input by the user and that no rule has been able to determine a value for it; it can be '*unknown*' which means the user was prompted for the variable but did not know it; or it can have one or more '*values*'. The unique aspects of the system are that not only can the expert system continue to execute when variables are unknown, but when variables do have values, each value has a confidence factor associated with it. Figure 3 shows examples of variable formats.

In the variable format, the translation and prompt string are self-explanatory. Each variable also has a type associated with it, either 'single-valued', 'multi-valued', or 'yes/no'. The 'expect' field is a list of the possible values for that variable which the user can select when and if he/she is prompted for that variable. The 'updated_by' field is a list of rules which are able to determine values for that variable, while the 'used_by' field contains rules which require this variable in order to fire. (It is possible that in order for a rule to fire, a variable must be '*unknown*'). The 'help' field is the information displayed when the user presses the help key, requesting more information on the variable being prompted for. The 'certainty-factor-range' (CFR) is particular to this system, and can have a value of '*unknown*', '*positive*', or '*full*'. The CFR being '*unknown*' means that this is a possible input for that variable. If the CFR is '*positive*', the user can input degrees of confidence from 0 to 100 for each of the inputted values for that variable. Finally, if the CFR is '*full*' the user can input degrees of confidence from -100 to 100 which mean a range from being 100% certain the variable is *not* a specific value to being 100% certain that the variable *is* a specific value.

The confidence factors relay the confidence the user has in a certain value of the variable. This is very important since there is

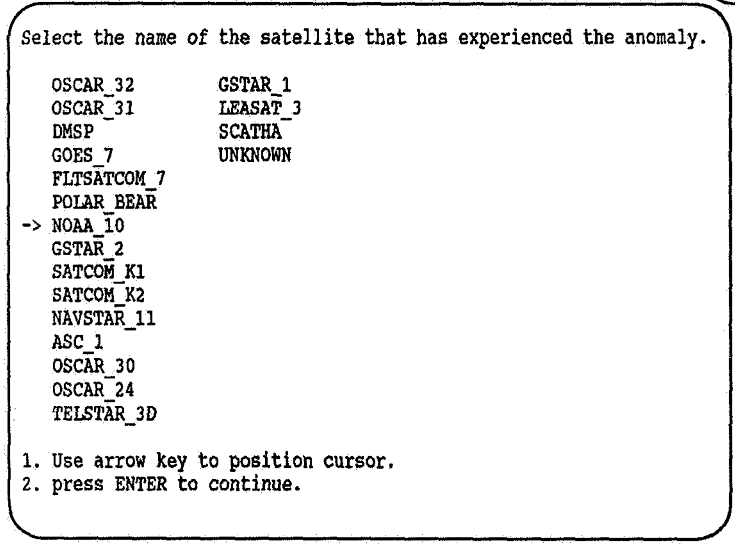


```
INCLINATION
===========
  TRANSLATION :: (the inclination of the plane of the orbit with respect
                   to the earth's equatorial plane )
  PROMPT :: (Select the inclination of the satellite with respect to the
                earth's equatorial plane. )
  TYPE :: SINGLEVALUED
  EXPECT :: (EQUATORIAL LOW_INCLINATION HIGH_INCLINATION POLAR OTHER)
  UPDATED-BY :: (RULE041 RULE133 RULE134 RULE135 RULE136 RULE132 RULE138
                  RULE139 RULE140 RULE141 RULE142 RULE137 )
  ANTECEDENT-BY :: (RULE026 RULE030)
  USED-BY :: (RULE017 RULE016 RULE091 RULE089)
  HELP :: ("Low inclination orbits are below 30 deg.  High
           inclination orbits are above 60 deg.  Polar orbits
           are above 80 deg.  Interplanetary orbits are undefined." )
  CERTAINTY-FACTOR-RANGE :: UNKNOWN


LT_RECUR
========
  TRANSLATION :: (the recurrence of an anomaly in a specific local-time
                   sector. )
  PROMPT :: ("Indicate the degree of certainty that you have that this
               type of anomaly has a strong tendency to recur in one local
               time sector, for example the nightside or the dayside of the
               earth?" )
  TYPE :: YES/NO
  USED-BY :: (RULE019 RULE020 RULE110 RULE054 RULE188 RULE189 RULE190
               RULE191 RULE192 RULE193 RULE194 RULE043 )
  HELP :: (The anomaly should have occurred a few times (i.e. six or more)
           before you have confidence that the recurrence is related to a
           specific local-time sector.  Generally we are asking if the
           anomaly has a very strong tendency to occur within a 12 hour range
           in local time. )
  CERTAINTY-FACTOR-RANGE :: POSITIVE
```

*Figure 3. Variable Format*

most likely information of which the user is not 100% sure. Such information is lost in normal programs. The combination of the confidence factors of variables and those of the rules propagates the confidence factors (or probabilities) to other variables which are determined by these rules and ultimately to the cause of the anomaly.

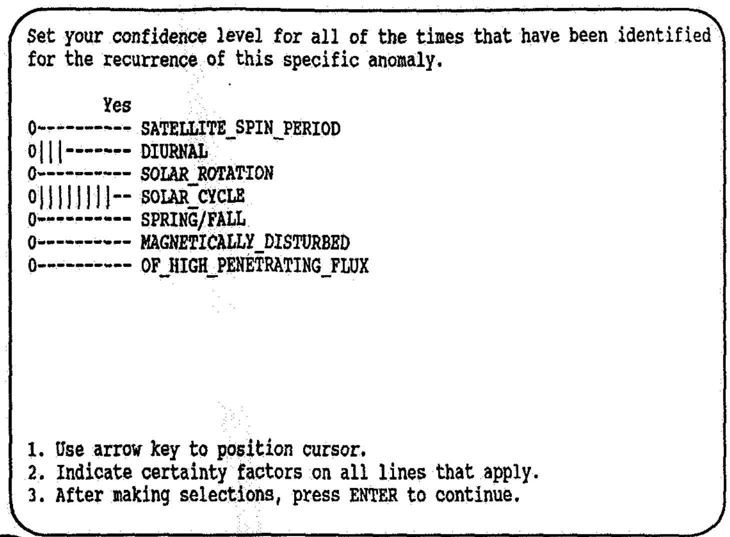
Figure 4 shows an input screen for a single-valued variable (which assumes 100% confidence — as are all inputs in normal programs), and a CFR of '*unknown*'.



```
Select the name of the satellite that has experienced the anomaly.

   OSCAR_32         GSTAR_1
   OSCAR_31         LEASAT_3
   DMSP             SCATHA
   GOES_7           UNKNOWN
   FLTSATCOM_7
   POLAR_BEAR
-> NOAA_10
   GSTAR_2
   SATCOM_K1
   SATCOM_K2
   NAVSTAR_11
   ASC_1
   OSCAR_30
   OSCAR_24
   TELSTAR_3D

1. Use arrow key to position cursor.
2. press ENTER to continue.
```

*Figure 4. Single-valued input*

Figure 5 is an example of the input screen for a multi-valued variable with a '*positive*' CFR. Notice how the variable in figure 5 can have more than one value, and each value has its own confidence factor associated with it.



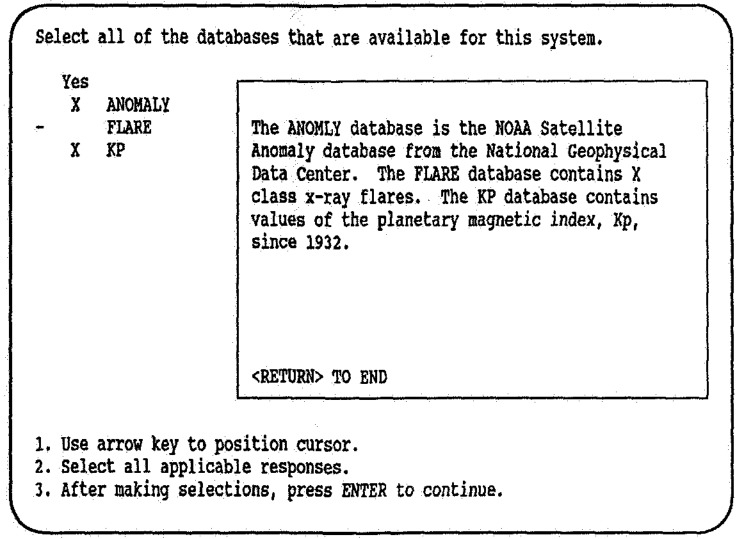
```
Set your confidence level for all of the times that have been identified
for the recurrence of this specific anomaly.

        Yes
0---------- SATELLITE_SPIN_PERIOD
0|||------- DIURNAL
0---------- SOLAR_ROTATION
0||||||||-- SOLAR_CYCLE
0---------- SPRING/FALL
0---------- MAGNETICALLY_DISTURBED
0---------- OF_HIGH_PENETRATING_FLUX




1. Use arrow key to position cursor.
2. Indicate certainty factors on all lines that apply.
3. After making selections, press ENTER to continue.
```

*Figure 5. Multi-valued input with confidence*

## FACT BASE

The fact base, a collection of informative sources related to the topic of interest, is the second basic part of an expert system. It can consist of as many separate data bases as may be deemed pertinent to solving the problem at hand. In the case of spacecraft anomalies, a fact base might contain information on the hardware currently in use, other active and past satellite systems, and historical data for orbital environments.

The database screen is shown in figure 6, which shows the databases available for this system along with an example of the expert system help facility which is available for any variable.



```
Select all of the databases that are available for this system.

  Yes
   X  ANOMALY
-     FLARE        The ANOMLY database is the NOAA Satellite
   X  KP           Anomaly database from the National Geophysical
                   Data Center.  The FLARE database contains X
                   class x-ray flares.  The KP database contains
                   values of the planetary magnetic index, Kp,
                   since 1932.



                   <RETURN> TO END

1. Use arrow key to position cursor.
2. Select all applicable responses.
3. After making selections, press ENTER to continue.
```

*Figure 6. Database selection screen*

An important advantage obtained in using the expert system is that once it has been established which databases are available, the rules determine which information is pertinent, access the database for the relevant information and apply this information, (all of which is transparent to the user). Also, the database accessing is modular and easily expandable, thus if more databases need to be added, only the selection screen needs to be changed, and the new rules added to the knowledge base.

These capabilities free the user from sifting through large amounts of data and ensure that only pertinent information and all pertinent information is used in the diagnosis.

## THE INTERFACE

The interface is one of the aspects which makes all expert systems different from one another. Since the expert shell, databases and knowledge base are independent and modular, the main purpose of the interface is to create a coordinating system which is not only user friendly, but also provides the necessary features to assist the user in understanding the system and the results.

### The Driver

The system's current interface driver translates forward chaining rules into a backward chaining sequence, prompting the user for information pertinent to the causes he/she wishes to consider. The main purpose of the driver is to maintain information regarding the variables which are being determined, the rules which can determine these variables, the status of the variables, and which rules can be fired.

Some variables are designated as initial variables or goal variables. The system first prompts the user for the initial variables. The driver then stacks the goal variables on the run time stack and searches the knowledge base for rules which determine (or 'update') these variables, and then puts them on the stack as well. The system focuses on those possibilities of high probability and then assists the user by directing him/her to areas of consideration that directly affect the particular problem. The goal (variable) in our system is the CAUSE of the anomaly, a multi-valued field variable with a *'full'* CFR, since it can take on any number of the four possible causes where each cause has its own confidence factor associated with it ranging from -100 to 100.

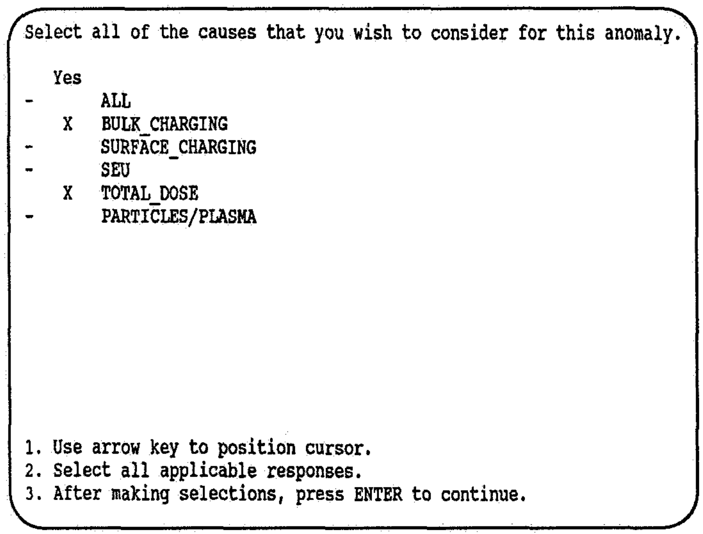
If a variable on the left hand side of a stacked rule is unset, this variable becomes the current goal variable and is put on the stack, and the process continues. If a variable is on the stack and has not determined by any rules, or by the available database, and it has a prompt string, the user is prompted for it. This can be thought of as a transformation of the forward chaining rules in the knowledge base into a backward chaining variable sequence. Once a variable has a value, it is removed from the stack and the rules which use this variable are fired, discarded, or require the driver to put the next variable on that rule's left hand side onto the stack. The chaining process continues until the stack is empty.

Any rule on the stack that can be fired does so transparently to the user, where the confidence factors of the individual variables on its left hand side (LHS) are used for determining the confidence or validity of the entire LHS. When a rule fires, it executes the right hand side (RHS), and the confidence factor associated with its LHS is used in conjunction with the confidence factor of the rule to propagate the confidence to the RHS. This

RHS execution can entail the setting of variables, the use of mathematical calculations, or the accessing of databases.

## Learning Tool

One of the most beneficial aspects of the system is its use as a learning tool for diagnosing spacecraft anomalies. A user is initially given a choice between either 'novice' or 'expert' mode for the current session. If the user selects the novice mode the system automatically gives detailed explanations and descriptions of terms and reasoning as the session progresses, in a sense teaching the user about the topic or topics. The expert mode, on the other hand, simply executes the session without giving these extra explanations, unless the user specifically requests them.



```
Select all of the causes that you wish to consider for this anomaly.

   Yes
 -        ALL
    X     BULK_CHARGING
 -        SURFACE_CHARGING
 -        SEU
    X     TOTAL_DOSE
 -        PARTICLES/PLASMA

1. Use arrow key to position cursor.
2. Select all applicable responses.
3. After making selections, press ENTER to continue.
```

*Figure 7. Causes selection screen*

The user is also given the option of selecting which causes are to be considered. (See figure 7.) This selection determines a knowledge base sub-group, so that only rules in this specific environmental area are considered. In this way the user can learn what variables, information and data affect, and are important to, that cause. In addition to this, in the features described next, the user is actually able to access the relevant rules him/herself and other variables and facts which were determined by using these rules.

## Features

The ability to add intricate features and options is primarily due to the modularity of the system which the expert shell and expert system knowledge base concept itself provide. These features are the most impressive in demonstrating the capabilities of the EnviroNET expert system and its advantages over the usual, strictly mathematical, programming techniques.

One feature which is vital for perceiving trends in environmental conditions which may have an effect on the satellite's anomalous behavior is graphics. This system has such capability.

For example, if the user inputs that one of the databases available is Kp, the system will ask if he/she wishes to see the Kp historical graph for the time around which the anomaly occurred. If the input is 'yes', then a graph similar to the one shown in figure 8 will be displayed. (If ,however, the date is '*unset*', then

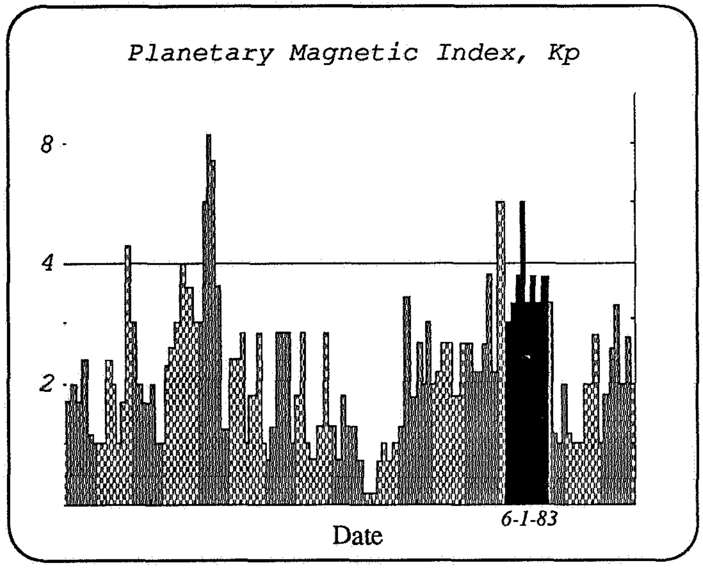
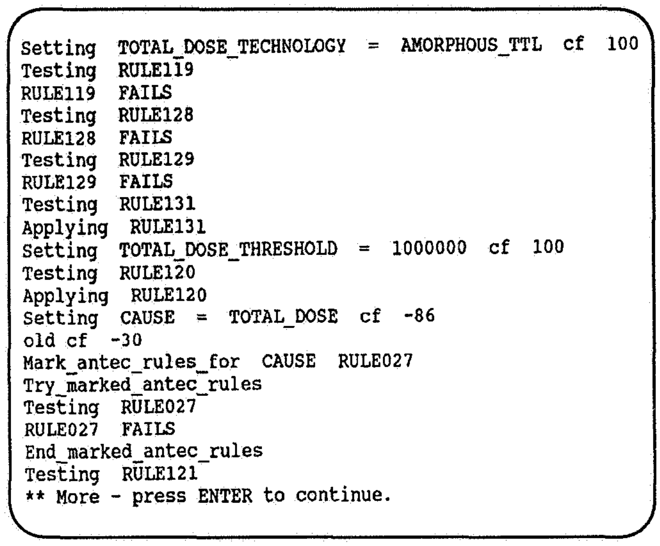


*Figure 8. Kp Graph*

the system will first ask for it, and if the date is '*unknown*' the system will ignore this line of questioning altogether.) This gives the user a much needed overall view of environmental information and conditions around the date in question.



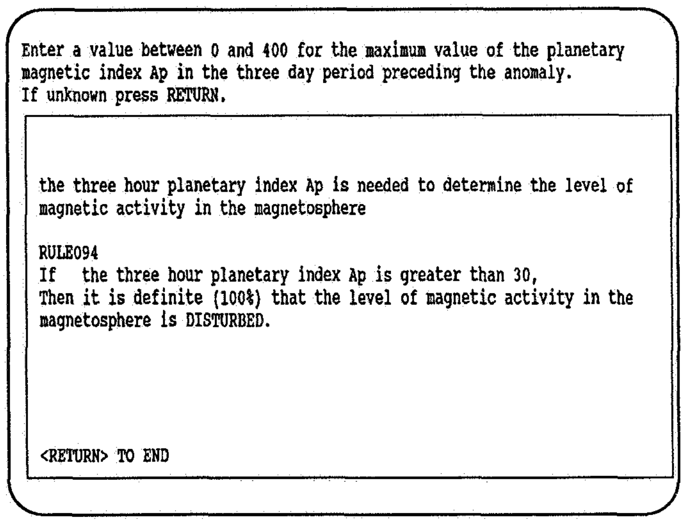
```
Setting  TOTAL_DOSE_TECHNOLOGY  =  AMORPHOUS_TTL  cf  100
Testing  RULE119
RULE119  FAILS
Testing  RULE128
RULE128  FAILS
Testing  RULE129
RULE129  FAILS
Testing  RULE131
Applying  RULE131
Setting  TOTAL_DOSE_THRESHOLD  =  1000000  cf  100
Testing  RULE120
Applying  RULE120
Setting  CAUSE  =  TOTAL_DOSE  cf  -86
old cf  -30
Mark_antec_rules_for  CAUSE  RULE027
Try_marked_antec_rules
Testing  RULE027
RULE027  FAILS
End_marked_antec_rules
Testing  RULE121
** More - press ENTER to continue.
```

*Figure 9. Trace example*

Another feature which makes the expert system unique is its trace capability. The user can turn on the trace and send it to the screen or a file. The trace shows the rules as they are tested, variables as they are pushed onto the runtime stack and determined, and searches of the databases. (See figure 9.) This allows the user to understand what is happening at any step and see the knowledge that is being used, thus giving the user confidence in the system. This type of capability is obviously not available in the purely algorithmic programs.

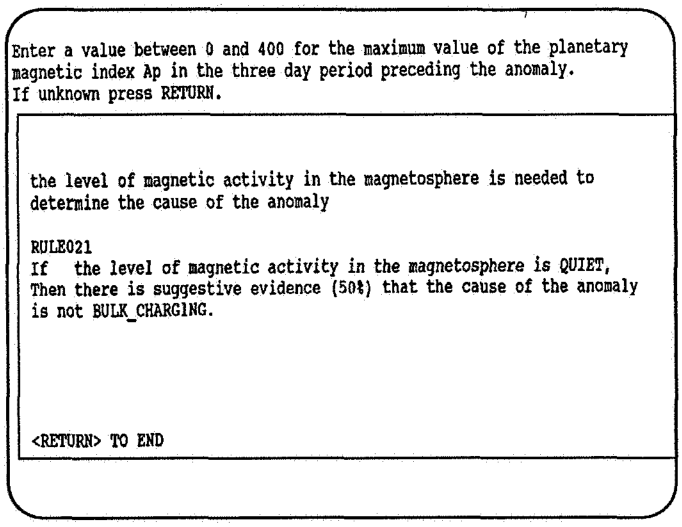
Due to the amount of information the user could be prompted for and depending on the particular session, the user may want to review his/her inputs. This capability is available in the 'REVIEW' facility. This option also provides the user with a simple way of comparing different inputs of different sessions.

A feature which demonstrates a definite advantage of the rule-based expert system is what is called the 'WHY' option. Any time the system prompts the user for a variable, the user can ask the expert system why the system needs this variable. The system then uses its run time stack (a backward chaining stack) to follow and show the reasoning backward to the goal; that is, the cause of the anomaly. Figures 10-11 show an example of this. This is not only vital to understanding and having confidence in the system, but it also is an important part of the expert system's use as a learning tool.



```
Enter a value between 0 and 400 for the maximum value of the planetary
magnetic index Ap in the three day period preceding the anomaly.
If unknown press RETURN.

the three hour planetary index Ap is needed to determine the level of
magnetic activity in the magnetosphere

RULE094
If   the three hour planetary index Ap is greater than 30,
Then it is definite (100%) that the level of magnetic activity in the
magnetosphere is DISTURBED.

<RETURN> TO END
```

*Figure 10. Backward reasoning*



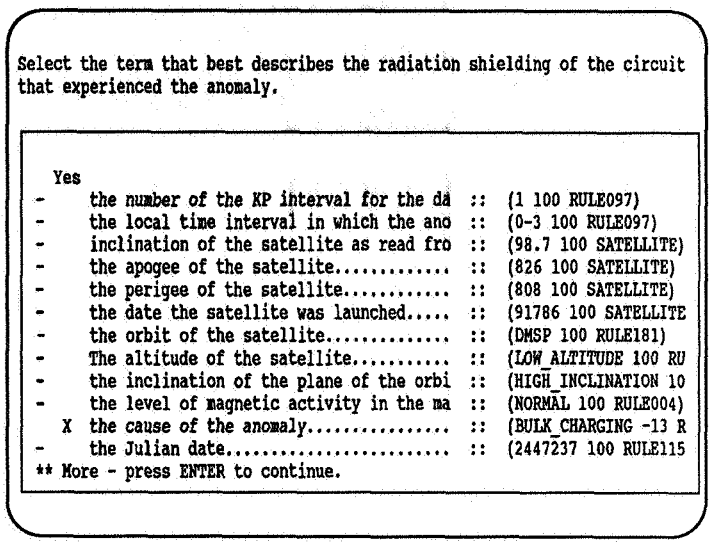
```
Enter a value between 0 and 400 for the maximum value of the planetary
magnetic index Ap in the three day period preceding the anomaly.
If unknown press RETURN.

the level of magnetic activity in the magnetosphere is needed to
determine the cause of the anomaly

RULE021
If   the level of magnetic activity in the magnetosphere is QUIET,
Then there is suggestive evidence (50%) that the cause of the anomaly
is not BULK_CHARGING.

<RETURN> TO END
```

*Figure 11. Backward reasoning (con't.)*

A final feature which sets the expert system apart is the 'HOW' command. As with all programs, the expert system is constantly determining variables by means other than the user inputting them, whether by the heuristics and algorithms in the rules or by extracting values from the databases. This command allows the users to, at any time, see what variables have been determined by means other than user input, their values, and which rules (or databases) were used to determine them. Figures 12-14 show an example of this feature. The user first selects which variables he/she wants to look at and then the system proceeds to show which rules determined them. Notice how it is possible for variables to be determined (or updated) by more than one rule. The user of course can choose any number of variables, though for this example only one variable, the cause of the anomaly, was selected.

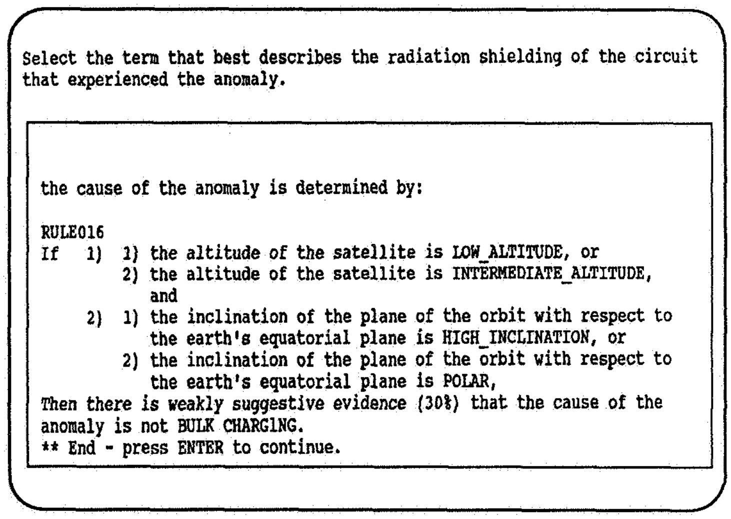
This feature not only gives the user complete control over the system, but allows him/her to see all the facts and knowledge that can be inferred from the inputs they have given, the available databases, and the expertise in the rules.

As a final option, the user is also allowed, at any point, to exit from the program or begin a new session without ever leaving the program's window screen.



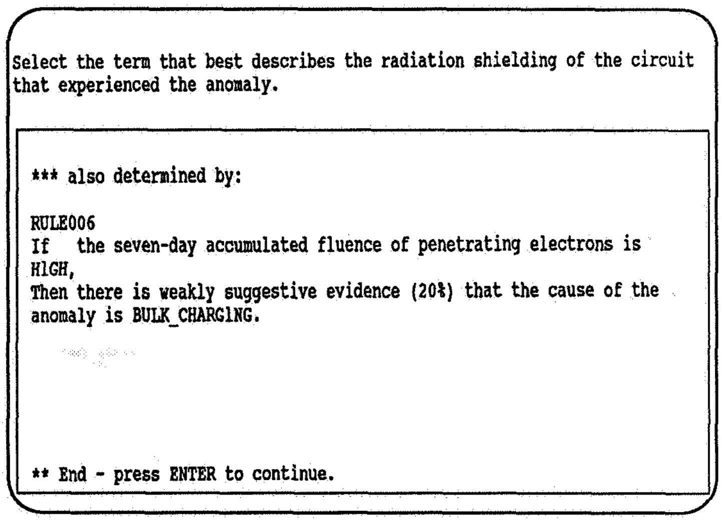
```
Select the term that best describes the radiation shielding of the circuit
that experienced the anomaly.

  Yes
-     the number of the KP interval for the da  ::  (1 100 RULE097)
-     the local time interval in which the ano  ::  (0-3 100 RULE097)
-     inclination of the satellite as read fro  ::  (98.7 100 SATELLITE)
-     the apogee of the satellite............  ::  (826 100 SATELLITE)
-     the perigee of the satellite...........  ::  (808 100 SATELLITE)
-     the date the satellite was launched.....  ::  (91786 100 SATELLITE
-     the orbit of the satellite.............  ::  (DMSP 100 RULE181)
-     The altitude of the satellite..........  ::  (LOW_ALTITUDE 100 RU
-     the inclination of the plane of the orbi  ::  (HIGH_INCLINATION 10
-     the level of magnetic activity in the ma  ::  (NORMAL 100 RULE004)
   X  the cause of the anomaly...............  ::  (BULK_CHARGING -13 R
-     the Julian date........................  ::  (2447237 100 RULE115
** More - press ENTER to continue.
```

*Figure 12. HOW facility*



```
Select the term that best describes the radiation shielding of the circuit
that experienced the anomaly.

the cause of the anomaly is determined by:

RULE016
If   1)  1) the altitude of the satellite is LOW_ALTITUDE, or
         2) the altitude of the satellite is INTERMEDIATE_ALTITUDE,
            and
     2)  1) the inclination of the plane of the orbit with respect to
            the earth's equatorial plane is HIGH_INCLINATION, or
         2) the inclination of the plane of the orbit with respect to
            the earth's equatorial plane is POLAR,
Then there is weakly suggestive evidence (30%) that the cause of the
anomaly is not BULK CHARG1NG.
** End - press ENTER to continue.
```

*Figure 13. HOW facility (con't.)*

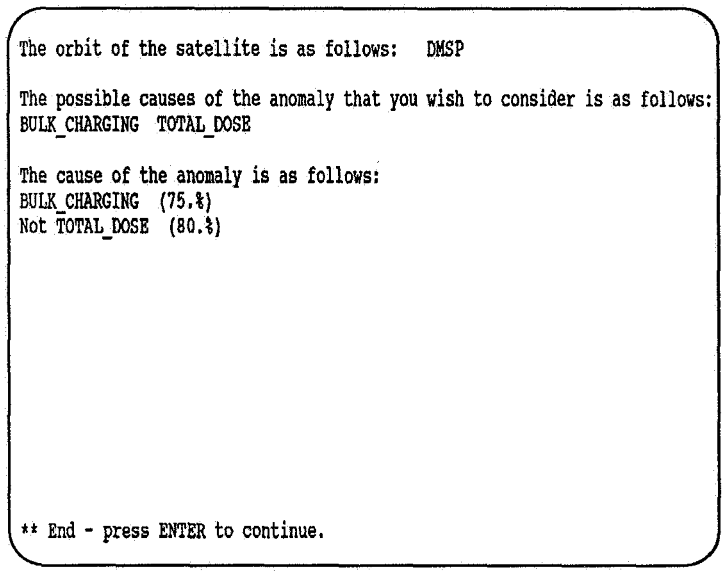


```
Select the term that best describes the radiation shielding of the circuit
that experienced the anomaly.

*** also determined by:

RULE006
If   the seven-day accumulated fluence of penetrating electrons is
H1GH,
Then there is weakly suggestive evidence (20%) that the cause of the
anomaly is BULK_CHARG1NG.

** End - press ENTER to continue.
```

*Figure 14. HOW facility (con't.)*

## RESULTS

The diagnostic results are in the form of probabilities due to both the confidence assigned to rules by the experts and also the confidence of variables input by the user. Both the rule/heuristic probabilities and the input of certainty/confidence factors are needed to diagnose anomalies as they contain vital knowledge which can only be represented as such. The results window is shown in figure 15.



```
The orbit of the satellite is as follows:   DMSP

The possible causes of the anomaly that you wish to consider is as follows:
BULK_CHARGING  TOTAL_DOSE

The cause of the anomaly is as follows:
BULK_CHARGING  (75.%)
Not TOTAL_DOSE  (80.%)









** End - press ENTER to continue.
```

*Figure 15. Results screen*

The results window in our system includes, in addition to the cause(s) of the anomaly, the orbit of the satellite, whether input by the user or determined by rules, and a list of the causes considered in the diagnostics. The window can easily be modified to display any other information which is considered important. In the example, the cause of the anomaly was determined to be bulk charging with a confidence of 75%, and determined *not* to be total radiation dose with a confidence of 80%. The knowledge base does, of course, contain rules and formulae which can determine the cause of the anomaly with 100% confidence, or completely rule out a particular cause. For these situations the system will simply say that the cause, for example, *is* bulk charging or *is not* total dose.

## CONCLUDING REMARKS

The expert system currently contains rules to diagnose anomalies resulting from surface charging, bulk charging, single event upsets, and total radiation. The system is extremely flexible; it can be easily expanded to include other causes as soon as the rules are obtained by simply adding these rules to the knowledge base. It can also obtain results even when information is not known.

The main concern with the system is the actual confidence and validity of the rules themselves. Since experts in any field are likely to disagree over certain areas, there may be rules to which other experts would apply slightly higher or lower degrees of confidence. This is certainly a consideration when using such a system, though it must be remembered that it is due to such a confidence/certainty question in the field that this type of expert system is needed. In addition, the features provided by the interface allow the user to see exactly what rules are being used so there is complete awareness and understanding of the formulae and knowledge being used.

Another reason this particular system is extraordinary is that its interface is completely general. Not only can the system run on many machines, the interface can be used in any field since the rules and knowledge base are completely independent of it. By substituting rules from another field, the system becomes an expert system for that field able to diagnose or solve problems towards which its tailored rules converge. In this sense the software is completely reusable and thus extends greatly beyond the scope of algorithmic programs.

The EnviroNET expert system combines the algorithmic capabilities of mathematical programs and diagnostic models with expert heuristic knowledge, and uses confidence factors in variables and rules to calculate probabilistic results. Since the causes of environmentally induced spacecraft anomalies depend not only on algorithms, but also on environmental conditions, rules and information which can rarely be known with 100% certainty, this new expert system technique which incorporates the human aspects of probabilities and expert knowledge is possibly the most comprehensive and thorough method for doing such diagnostics.

## REFERENCES

1. Giarratoano, J. C. (1989, May). *CLIPS User's Guide, Version 4.3 of CLIPS*. Artificial Intelligence Section, Lyndon B. Johnson Space Center.

2. Koons, H. C., & Gorney, D. J. (1988, Dec). *Spacecraft Environmental Anomalies Expert System: A Status Report*. Space Sciences Laboratory, The Aerospace Corporation.

3. Lauriente, M. (1989, May). *Proceedings: Environmentally Induced Spacecraft Anomalies Workshop*. Code 410.1, NASA/GSFC, Greenbelt, MD.

4. Private Communication with Lyndon B. Johnson Space Center AI Laboratory

5. *The EnviroNET User Guide*. Code 410.1, NASA/GSFC, Greenbelt, MD.

# *Robotics/Intelligent Control*

# A Hierarchical Distributed Control Model for Coordinating Intelligent Systems

Richard M. Adler
Symbiotics, Inc.
875 Main Street
Cambridge, MA 02139

## ABSTRACT

This paper describes a hierarchical distributed control (HDC) model for coordinating cooperative problem-solving among intelligent systems. The model was implemented using SOCIAL, an innovative object-oriented tool for integrating heterogeneous, distributed software systems. SOCIAL embeds applications in "wrapper" objects called Agents, which supply predefined capabilities for distributed communication, control, data specification and translation. The HDC model is realized in SOCIAL as a "Manager" Agent that coordinates interactions among application Agents. The HDC-Manager: indexes the capabilities of application Agents; routes request messages to suitable server Agents; and stores results in a commonly accessible "Bulletin-Board". This centralized control model is illustrated in a fault diagnosis application for launch operations support of the Space Shuttle fleet at NASA, Kennedy Space Center.

**Keywords:** distributed artificial intelligence, systems integration, hierarchical distributed control, intelligent control, cooperative problem-solving

## INTRODUCTION

Knowledge-based systems are helping to automate important functions in complex problem domains such as operations and decision support. Successful deployment of intelligent systems requires: (a) integration with existing, conventional software programs and data stores; and (b) coordinating with one another to share complementary knowledge and skills, much as people work together cooperatively on related tasks. These requirements are difficult to satisfy given existing AI technologies. Current knowledge-based systems are generally single-user, standalone systems based on heterogeneous data and knowledge models, development languages and tool shells, and processing platforms. Interfaces to users, databases, and other conventional software systems are typically custom-built and difficult to adapt or interconnect. Moreover, intelligent systems developed independently of one another tend to be ignorant of information resources, problem-solving capabilities, and access protocols for peer systems.

SOCIAL is an innovative collection of object-oriented tools designed to alleviate these pervasive integration problems [Ad90b]. SOCIAL provides a family of "wrapper" objects, called *Agents*, that supply predefined capabilities for distributed communication, control, data specification and translation. Developers embed programs within Agents, using high-level, message-based interfaces to specify interactions between programs, their embedding Agents, and other application Agents. The relevant Agents transparently manage the transport and mapping of specified data across networks of disparate processing platforms, languages, development tools, and applications. SOCIAL's partitioning of generic and application-specific behaviors shields developers from network protocols and other low-level complexities of distributed computing. More important, the interfaces between applications and SOCIAL Agents are modular and nonintrusive, minimizing the number, extent, and cost of modifications necessary to re-engineer existing systems for integration. Non-intrusiveness is particularly important in mission-critical space and military applications, where alterations for integration entail stringent validation and verification testing.

This paper focuses on a specialized "Manager" Agent that realizes a hierarchical distributed control (HDC) model on top of SOCIAL's basic integration services. The SOCIAL HDC-Manager Agent coordinates the activities of Agents that embed independent knowledge-based and conventional applications relating to a common domain such as decision or operations support. Such centralized control models are important for managing distributed systems that evolve over time through the addition of new applications and functions. Centralized control is also important for organizing complex distributed systems that display not only small-scale, one-to-one relationships, but also large-scale structure, such as clustering of closely related subsets of application elements.

The HDC-Manager Agent's coordination functionality derives from a set of centralized control services including: maintaining an index knowledge base of the capabilities, addresses, and access message formats for application Agents; formatting and routing requests for data or problem-solving processing to suitable server Agents; and posting request responses and other globally useful data to a commonly accessible "Bulletin-Board".

The next section of the paper reviews the overall architecture and functionality of SOCIAL. Subsequent sections describe the structure and behavior of the HDC-Manager Agent and illustrate its application in the domain of launch operations support for the Space Shuttle fleet at NASA, Kennedy Space Center. Specifically, a HDC-Manager Agent coordinates the activities of standalone expert systems that monitor and isolate faults in Shuttle vehicle and Ground Support systems. The cooperative problem-solving enabled by the HDC-Manager produces diagnostic conclusions that the applications are incapable of reaching individually.

## OVERVIEW OF SOCIAL

The central problems of integrating heterogeneous distributed systems include:

- communicating across a distributed network of heterogeneous computers and operating systems in the absence of uniform interprocess communication services;
- specifying and translating information (i.e., data, knowledge, commands), across applications, programming languages and development shells with incompatible native data representations;
- coordinating problems-solving across applications and development tools that rely on different internal models for communication and control.

SOCIAL addresses these issues through a unified collection of object-oriented tools for distributed communication, control, data (and data type) specification and management. Developers access the services provided by each tool through high-level Application Programming Interfaces (APIs). The APIs conceal the low-level complexities of implementing distributed computing systems. This means that distributed systems can be developed by programmers who lack expertise in areas such as interprocess and network communication (e.g., Remote Procedure Calls, TCP/IP, ports and sockets), variations in data architectures across vendor computer platforms, and differences among data and control interfaces for standard development tools such as AI shells. Moreover, SOCIAL's high-level development interfaces to distributed services promote modularity, maintainability, extensibility, and portability.

The overall SOCIAL architecture is summarized in Figure .1. SOCIAL's predefined distributed processing functions are bundled together in objects called ***Agents***: Agents represent the active computational processes within a distributed system. Developers assemble distributed systems by:

(a) selecting and instantiating Agents from SOCIAL's library of predefined Agent classes; and (b) embedding individual application elements such as programs and databases within Agents. Embedding consists of using the APIs for accessing SOCIAL's distributed processing capabilities to establish the desired interactions between applications, their associated wrapper Agents, and other application Agents. New Agent subclasses can be created through a separate development interface by customized or combining services in novel ways to satisfy unique application requirements. These new Agent types can be incorporated into SOCIAL's Agent library for subsequent reuse or adaptation. The following subsections review the component distributed computing technologies used to construct SOCIAL Agents.

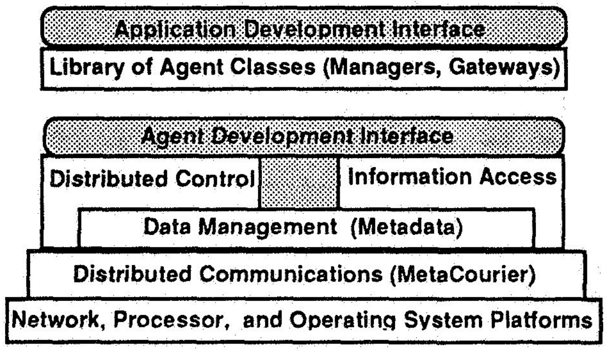

Application Development Interface
Library of Agent Classes (Managers, Gateways)
Agent Development Interface
Distributed Control
Information Access
Data Management (Metadata)
Distributed Communications (MetaCourier)
Network, Processor, and Operating System Platforms

Figure .1: Architecture of the SOCIAL Toolset

### Distributed Communication

SOCIAL's distributed computing utilities are organized in layers, enabling complex functions to be built up from simpler ones. The base or substrate layer of SOCIAL is the MetaCourier tool, which provides a high-level, modular distributed communications capability for passing information between applications based on heterogeneous languages, platforms, operating systems, networks, and network protocols [Sy90].

The Agent objects that integrate computer-based applications or resources are defined at SOCIAL's MetaCourier level. Developers use the MetaCourier API to pass messages between applications and their embedding Agents, as well as among application Agents. Messages typically consist of: commands that an Agent passes directly into its embedded application, such as database queries or calls to execute signal processing programs; data arguments to program commands that an Agent might call to invoke its embedded application; and symbolic flags or keywords that signal the Agent to invoke one or another fully preprogrammed interactions with its embedded application. For example, a high-level MetaCourier API call issued from a local LISP-based application Agent such as:

```
(Tell :agent 'sensor-monitor :sys 'Symb
'(poll measurement-Z))
```

transports the message contents, in this case a command to poll measurement-X, from the calling program to the Agent ***sensor-monitor*** resident on platform ***Symb1***. The Tell function initiates a message transaction based on an asynchronous communication model; the application Agent that issues such a message can immediately move on to other processing tasks. The MetaCourier API also provides a synchronous "Tell-and-Block" message function for "wait-and-see" processing models.

Agents contain two procedural methods that control the processing of messages, called in-filters and out-filters. In-filters parse incoming messages according to the argument list structure specified when the Agent is defined. After parsing the message, an in-filter typically either invokes the Agent's embedded resource or application, or passes the message (which it may modify) onto another Agent. The MetaCourier semantic model entails a directed acyclic computational graph of passed messages. When no further passes are required, the in-filter of the terminal Agent runs to completion. This Agent's out-filter method is then executed to prepare a message reply, which is automatically returned (and possibly modified) through the out-filters of intermediate Agents back to the originating Agent (i.e., the target Agent for the original Tell call).

A MetaCourier runtime kernel resides on each application host. The kernel provides: (a) a uniform message-passing interface across network platforms; and (b) a scheduler for managing messages and Agent processes (i.e., executing filter methods). Each Agent contains two attributes that specify associated Host and an Environment objects. These MetaCourier objects define particular hardware and software execution contexts for Agents, including the host processor type, operating system, network type and address, language compiler, linker, and editor. The MetaCourier kernel uses the Host and Environment associations to manage the hardware and software platform specific dependencies that arise in transporting messages between heterogeneous, distributed Agents (cf. Figure .2).

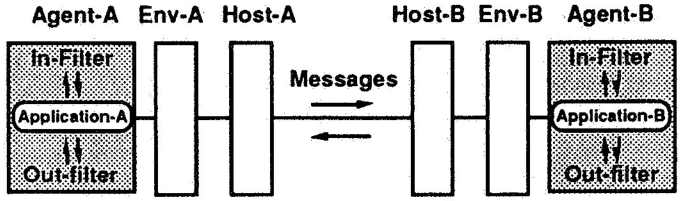


Figure .2: Operational Model of MetaCourier

MetaCourier's high-level message-based API is basically identical across different languages such as LISP or C. MetaCourier's communication model is also symmetrical or "peer-to-peer". In contrast, client-server models based on remote procedure call communication formalisms (RPCs) are asymmetric: only clients can initiate communication and while multiple clients can interact with a particular server, a specific client process can only interact with a particular server. Moreover, until recently, RPCs were restricted to inefficient synchronous (i.e., blocking) communications.

### Data Specification and Translation

A major difficulty in getting heterogeneous applications and information resources to interact with one another is the basic incompatibility of their underlying models for representing data, knowledge, and commands. These problems are compounded when applications are distributed across heterogeneous computing platforms with different data architectures (e.g., opposing byte ordering conventions).

SOCIAL's Metadata subsystem addresses these complex compatibility problems. MetaData provides an object-oriented data model for specifying and manipulating data and data types across disparate applications and platforms. Developers access these tools through a dedicated API. MetaData handles three basic tasks: (a) encoding and decoding basic data types (e.g., character, integer, float), transparently across different machine architectures; (b) describing data of arbitrarily complex abstract types (e.g., database records, frames, b-trees), in a uniform object-oriented model; and (c) mapping across data models native to particular applications and SOCIAL's generic data model. Like MetaCourier, MetaData's object-oriented API is basically uniform across different programming languages. Also, MetaData allows new types to be defined and manipulated dynamically at runtime. Most alternative data management tools, such as XDR, are static and non-object-oriented.

SOCIAL integrates MetaData with MetaCourier to obtain transparent distributed communication of complex data and data types across heterogeneous computer platforms as well as across disparate applications: developers embed MetaData API function calls within the in-filter and out-filter methods of interacting Agents, using MetaCourier messages to transport MetaData objects across applications residing on distributed hosts. MetaData API functions decode and encode message contents, mapping information to and from the native representational models of source and target applications and MetaData objects. SOCIAL thereby separates distributed communication from data specification and translation, and cleanly partitions both kinds of generic functionality from application-specific processing.

### Distributed Control and Information Access

SOCIAL's third layer of object-oriented tools establishes custom, high-level API interfaces for Agent classes specialized for particular integration or coordination functionality. Lower-level MetaCourier and MetaData API functions are used to construct these Agent API data and control interfaces. The high-level APIs largely conceal MetaCourier and MetaData interfaces from SOCIAL users. Thus, developers of distributed systems typically use the predefined specialized Agent classes as the top-level building blocks for satisfying their particular architectural requirements, accessing the functionality of each such Agent type through the dedicated high-level API interfaces.

For example, applications and data stores are often constructed using standard development tools such as database management systems (DBMSs) and AI shells. SOCIAL Database and Knowledge ***Gateway*** Agent classes abstract and isolate the application-independent aspects of the control and data interfaces to such shells as generic, reusable API interfaces. Similarly, ***Managers*** are specialized Agents for coordinating applications integrated via Gateways and other kinds of SOCIAL Agents to work together cooperatively. The HDC-Manager, described in the following sections, defines one kind of centralized model for distributed control of application Agents. If necessary, developers can access SOCIAL's various tool layers to modify or extend existing Agent APIs and define corresponding new subclasses of Gateway, Manager, or fully custom Agents.

Gateway and Manager Agent APIs depend heavily on MetaCourier and MetaData capabilities for controlling and manipulating messages. For instance, the root Knowledge Gateway Agent class defines standard MetaCourier in-filter and out-filter methods. The generic in-filter parses and processes messages that represent either: (a) incoming requests initiated by other application Agents; or (b) requests initiated by the Agent's embedded application to pass on to other application Agents. Similarly, the generic out-filter either: (a) assembles responses to in-coming requests; or (b) relays replies to prior out-going requests back to the embedded application. All subclasses of Knowledge Gateway Agents inherit these generic shell- and application-independent control behaviors.

SOCIAL uses MetaData to define a uniform representational model for data and knowledge structures such as relational tuples, facts, fact-groups, rules, and frames. Subclasses of the root Knowledge Gateway Agent class define API functions that map transparently between SOCIAL's canonical representation and the knowledge models native to particular AI development tools. For example, CLIPS Knowledge Gateway Agents translate between MetaData frames and CLIPS deftemplates, while KEE Gateway Agents map between MetaData frames and KEE unit objects. MetaData objects are also used to pass commands to AI shells associated with Gateway Agent subclasses (e.g., to reset a fact base, execute a rule-based inference engine, or load application files). This uniform mapping approach simplifies the problem of interconnecting N disparate systems from O(N*N) to O(N).

Developers integrate a shell-based application by embedding it in an instance of the relevant Gateway Agent subclass, using its shell-specific API functions to program the required interactions: the API calls specify the particular data or commands to be injected into, or extracted from, the embedded application's knowledge bases through the shell's control interfaces. The strategy of modular, specialized interface functionality used in SOCIAL's predefined Gateway Agent classes is directly applicable for designing custom types of Agent for integrating standalone applications or systems implemented using in-house, proprietary development shells.

## THE HDC-MANAGER AGENT

Taken together, the SOCIAL tools described thus far - MetaCourier, MetaData, and Gateway Agents - provide adequate support for integrating conventional heterogeneous software systems. The interactions in such systems are relatively few in number and can be prescribed in a fixed, determinate order. For example, satellite telemetry data can be fed to Agents embedding signal processing and pattern recognition programs to clean and filter data and check for significant events. Database Gateway Agents would be used to dump all data to archival storage, while extracting and writing noteworthy events to on-line storage systems. Scientists might then study the data through Agents that embed data analysis and visualization programs, possibly through an intermediary User Interface Agent. The developer of such a system would embed the constituent applications and databases in suitable Agents and specify their direct, one-to-one interactions in terms of the relevant Agent APIs.

Once autonomous knowlege-based systems are incorporated into a distributed system, integration tools alone are no longer fully adequate. First, the sequencing or composition of behaviors both within and across autonomous systems is typically determined dynamically, based on the content of incoming data at run-time. A decentralized approach to managing such data-driven behaviors quickly becomes intractable. The problem is particularly acute in distributed systems that evolve over an extended lifecycle, in which application elements are enhanced, added, superseded, or reorganized (i.e., broken apart or consolidated), over time.

Second, complex organizational relationships emerge among clusters of applications in large-scale distributed intelligent systems. For example, programs that automate on-line operations of computer networks and other complex systems are naturally coupled more closely to one another than to decision or maintenance support tools for the same target domain. At the same time, interactions regularly take place between applications across functional groupings. For instance, planning and scheduling tools (decision support) dictate system configuration activities (operations support), while behavioral anomalies detected in on-line systems (operations support) trigger and guide troubleshooting, diagnosis, and repair activities (maintenance support). Ideally, interfaces for such cross-group interactions should be designed at the cluster level, to minimize sensitivity to modifications within functional groups.

In short, the intelligent application elements of complex distributed systems must not only be integrated, but also *coordinated*. Systematic coordination is necessary both to manage large numbers of dynamically evolving interaction pathways and to capture complex logical relationships among functional elements. The HDC-Manager is the first specialized class of SOCIAL Agents developed to address these organizational requirements.

### HDC-Manager Functionality

In essence, the HDC model performs the same role in a complex distributed system played by a human manager in a large organization. The HDC-Manager is associated with one or more application Agents, called its *subordinates*. The HDC-Manager's specific functions or responsibilities with respect to these other Agents include:

- providing a centralized Index knowledge base that specifies: each available information source or problem-solving service; the subordinate Agent that can provide that resource; the Agent's logical address; and a procedure for converting data in a generic resource request into a message format that is suitable for that server Agent;
- analyzing tasks requesting information or problem-solving services based on the Index knowledge base and routing suitable messages to the relevant subordinate Agents to accomplish those tasks;
- mediating all interactions with external (i.e., non-subordinate) Agents;
- providing a centralized Bulletin-Board to store problem-solving results and other data of common utility for shared access by subordinate and external Agents;

The HDC-Manager realizes the advantages of a centralized control architecture in a complex distributed system:

- efficient *global* coordination of individual problem-solving Agents;
- modularity;

- extensibility and maintainability;
- support for heterogeneity.

Modularity derives from the HDC-Manager's centralized Bulletin-Board and Index knowledge base. Each Index entry identifies services available from subordinate Agents symbolically (e.g., find-fault-precedents). Each Bulletin-Board posting identifies the item type, such as service request or reply, the posting Agent, and the requesting Agent (when appropriate). Subordinate Agents do not need to know about the functionality, structure, or even the existence of any other application Agents; they only require: (a) the generic high-level API interface to the HDC-Manager Agent; and (b) knowledge of the symbolic names used by the HDC-Manager to index the resource types available within a specific distributed application. The same minimal requirements hold for external application and Manager Agents that need to interact with an HDC-Manager to obtain information or services from its subordinates.

Extensibility and maintainability follow from the HDC-Manager's modular architecture: new subordinate Agents are incorporated simply by: (a) updating the Index knowledge base with appropriate entries to describe its services; and (b) extending other subordinate Agents, as needed, to be able to request new capabilities and process the results. Moreover, existing subordinate Agents can be reconfigured with minimal disruption. For example, suppose that problem-solving functions in one subordinate Agent are reallocated to some other application Agent, old or new. Neither the Manager's other subordinates nor any external Agents have to be modified; only the HDC-Manager Index knowledge base needs to be updated to reflect the configuration changes.

Heterogeneity follows from the modular nature of SOCIAL Agents in general. The high-level API to the HDC-Manager Agent's coordination capabilities is distinct from, but fully compatible with, the API interfaces used to embed applications with Gateways or other kinds of SOCIAL Agents. Thus, the HDC-Manager Agent operates transparently with respect to the physical distribution and internal architectures (i.e., communication, control, and knowledge structures), of subordinate and external Agents with which it interacts. Consequently, an HDC-Manager can subordinate standalone application Agents, Gateway Agents, and other Manager Agents, HDC or otherwise, with equal ease. In particular, HDC-Manager Agents can be organized in a nested hierarchy to support large complex distributed systems, as illustrated in Figure .3.

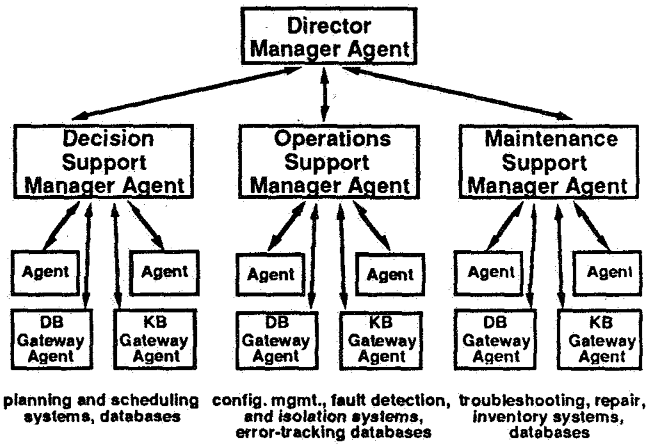

Director Manager Agent
Decision Support Manager Agent
Operations Support Manager Agent
Maintenance Support Manager Agent
Agent
Agent
Agent
Agent
Agent
Agent
DB Gateway Agent
KB Gateway Agent
DB Gateway Agent
KB Gateway Agent
DB Gateway Agent
KB Gateway Agent
planning and scheduling systems, databases
config. mgmt., fault detection, and isolation systems, error-tracking databases
troubleshooting, repair, inventory systems, databases

Figure .3: Hierarchy of HDC-Manager Agents

### HDC-Manager Architecture and Operation

The HDC-Manager Agent was implemented using SOCIAL and Common Lisp in a uniform, fully object-oriented manner. The Agent is comprised of a collection of state variables and utility methods (cf. Figure .4). The value for each state variable consists of a list of MetaData objects, such as Agent-Index-Items. Each such object type has associated test predicate, instance creation, and access functions (e.g., Agent-Index-ItemP, Create-Agent-Index-Item, Check-Agent-Index). These low-level functions, written using the MetaData API, are invoked through a higher-level HDC-Manager API and are transparent to developers. Five state variables store the static and dynamic information necessary for the HDC-Manager to function:

- a Task-Agenda for posting, prioritizing, and dispatching service request Tasks;
- a Bulletin-Board for posting Task results and other data items to a common memory store accessible to all subordinate and external Agents;
- an Index knowledge base that enumerates subordinate Agents, their logical locations, server capabilities, and functions for assembling data into suitable command message formats;
- a set of Prioritization Conditions for ordering the Agenda queue of pending Tasks to be routed by the Manager;
- an Activity Log for tracing and debugging Manager behavior during application development;

**State Variables**

**Task Agenda**<br>
**Bulletin-Board**<br>
**Index of subordinate Agents**<br>
**Task Priorities**<br>
**Action Log**

**Procedural Methods**

**MetaCourier Methods:**
  **in-filter, out-filter**
**Utility Methods:**
  **HDC control model methods**
  **access methods for Manager state variables**
**Application-Specific Methods:**
  **local Manager task handlers, ext. interfaces**

Figure .4: HDC-Manager Agent Structures

Currently, the Index knowledge base and the Prioritization conditions represent static structures that are specified once, when an HDC-Manager Agent is first defined to coordinate a specific set of applications. Typically, changes are made during development, but infrequently thereafter, when subordinate application Agents are added, removed, or restructured. (Self-adapting systems could modify Priority Conditions or even create new server Agents dynamically; however, we have not yet investigated such possibilities.) The remaining three state variables are more dynamic structures: their contents change continually as the HDC-Manager regulates an operational distributed system.

The HDC-Manager Agent also incorporates four sets of supporting procedural methods:

- in-filter and out-filter methods for parsing and responding to incoming MetaCourier messages and processing replies to outgoing messages;
- auxiliary API utility methods that realize the HDC-Manager's global hierarchical control model;
- auxiliary API utility methods for creating and modifying HDC-Manager data items, posting them to and retrieving them from the Agent's state variables;
- optional methods for handling application Tasks within the HDC-Manager itself. Such methods may be called for when a Manager Agent is configured as a subordinate to other Manager Agents.

The HDC-Manager's specialized coordination functions and information structures are accessed through a dedicated API (cf. Figure .5). This API is implemented via lower-level MetaData and MetaCourier capabilities and APIs. The high-level API shields SOCIAL developers from the underlying mechanics of packaging, transporting, and deciphering messages containing HDC-Manager commands and data structures among heterogeneous Agents. The API utility methods for accessing HDC-Manager state variables and their contents include:

- two methods for searching the Index Knowledge Base and Bulletin-Board. Both methods call a generic symbolic pattern-matching function with application-specific search conditions;
- a single Create-Item method, which dispatches to the various functions that create instances

of HDC-Manager MetaData object types: Index and Bulletin-Board entry items; Priority-Conditions, and Tasks (Service Requests);

- four parallel methods for modifying HDC-Manager state variables: Task-Agenda, Index Knowledge Base, Bulletin-Board, and Priority-Conditions; Each method supports keyword options for resetting the variables, posting and deleting specific data items;

The HDC-Manager API is also used to invoke the utility methods that implement the HDC control model. These methods are generally triggered automatically, from the Manager's predefined infilter and out-filter, but can be activated by other Agents as required. HDC-Manager control methods include:

- a Command-Manager method, which dispatches API command messages, either to internal HDC-Manager API methods *or* to the Command-Manager of another HDC-Manager Agent. This method also traps illegal commands;

- an Initialize method for resetting the HDC-Manager state variables and performing any application-specific actions;

- a Prioritize-Agenda method for sorting pending Tasks for the Manager to dispatch in accordance with the declarative ordering conditions specified in the Priorities-Conditions state variable. Each condition specifies a Task Attribute (e.g., service-category, priority), and an optional list of Attribute values for ordinal sorting;

- a Task-Dispatcher method, which uses the Index Knowledge Base to generate and send a suitable command message to the relevant subordinate Agent requesting the service specified in a Task;

- a top-level Control-Cycle method that invokes the Prioritize-Agenda and Task-Dispatcher methods that ground the HDC model;

- a Log-Utility method. A menu-driven trace/debug facility can be used to toggle the Activity Log, which tracks all messages processed by the Command-Manager, and other flags that trace of all runtime modifications to Manager state variables.

| **Control Methods** | **Data Structure Accessors** |
|---|---|
| **Command-Manager** | **Check-Agent-Index** |
| **Control-Cycle** | **Check-Bulletin-Board** |
| **Initialize** | **Create-Item** |
| **Log-Utility** | **Modify-Agenda** |
| **Prioritize-Agenda** | **Modify-Agent-Index** |
| **Task-Dispatcher** | **Modify-Bulletin-Board** |
|  | **Modify-Priority-Conditions** |

Figure .5: HDC-Manager Agent Utility Methods

The Command-Manager enforces regulated hierarchical control channels. A subordinate Agent can communicate with any HDC-Manager Agent within a Manager hierarchy; however, any such message is processed first by the Agent's immediate superior and then by all intervening Managers. This design makes it possible to define and enforce alternative organizational reporting policies. The default policy is very flexible: messages are tracked, but passed along the Manager hierarchy without filtering. Thus, any subordinate Agent can access the Bulletin-Board or request services from other Managers through its immediate Manager. More or less restrictive control architectures may be appropriate under different conditions. For example, messages from subordinates to higher-level Managers in time-critical applications could be screened or prioritized. Parallel, antagonistic or competitive models can also be explored. Finally, the HDC-Manager architecture does not preclude a subordinate reporting to multiple Managers, thus permitting non-hierarchical models (e.g, matrix structures), or elaborate hybrid organizational structures.

The HDC-Manager was designed in a modular fashion to facilitate maintenance and extension. API accessor utility methods conform to uniform conventions for naming, argument call structures, and parallel behavior. For example, Tasks or other data structures can be modified by adding attributes (e.g., timestamps for First-In-First-Out ordering), or changing attribute names. The developer merely changes the relevant Create-X function (and possibly one of the API Check methods). New HDC-Manager state variables and data objects to populate them can be added by implementing appropriate MetaData type-checking predicates, creation and accessor functions, adding a case to the Create-Item API method, and extending the table that drives the Command-Manager and Task-Dispatcher control methods.

Similarly, the HDC-Manager API control methods share parallel argument call structures and can be modified or extended selectively. For example, the Initialize method can be customized to perform any actions required to load and initiate all subordinate Agents and their embedded applications. Moreover, as noted above, alternative organizational policies can be implemented to capture logical relationships specific to particular distributed systems. All such modifications are implementing by creating HDC-Manager Agent subclasses with custom methods that override the standard methods defined by the root Agent class. For instance, the default initialization behavior can be redefined simply by creating a subclass Agent with a new Initialize method that calls a Custom-Initialize function. Specialization preserves the structure of the original HDC-Manager Agent class for use in applications where it is suitable. At the same time, inheritance and functional abstraction (through method dispatching) promotes adaptability and compact definitions for customized Manager Agent subclasses.

## USING THE HDC-MANAGER AGENT FOR LAUNCH OPERATIONS SUPPORT

Over the past decade, NASA Kennedy Space Center (KSC) has developed knowledge-based systems to increase automation of operations support tasks for the Space Shuttle fleet. Major applications include, operations support of the Shuttle Launch Processing System, monitoring, control, fault isolation and management of on-board Shuttle systems and Ground Support Equipment. Prototypes have been tested successfully (off-line) in support of several Shuttle missions and are currently being extended and refined for formal field testing and validation. Final deployment will require integrating these applications, both with one another and with existing Shuttle operations support systems.

KSC recently initiated the EXODUS project (Expert Systems for Operations Distributed Users) to prepare for this challenging systems integration task. As part of this effort, KSC is funding Symbiotics, Inc. to develop the SOCIAL toolset to help validate, refine, and ultimately implement the proposed EXODUS architecture [Ad90a]. Proof-of-concept prototypes have been constructed to demonstrate central EXODUS design concepts: distributed data transfer; non-intrusive physical distribution of knowledge bases from existing intelligent system to facilitate resource control and sharing; and integration of expert systems and databases via Gateway Agents for CLIPS, KEE, and Oracle development tools. This section describes a fourth EXODUS prototype, which used a SOCIAL HDC-Manager Agent to coordinate the fault isolation activities of two previously standalone intelligent systems.

### Background on KSC Launch Operations

Processing, testing, and launching of Shuttle vehicles takes place at facilities dispersed across the KSC complex. Many activities, such as storing and loading fuels and controlling the environments of Shuttles on Launch Pads require elaborate electromechanical Ground Support Equipment. The Launch Processing System (LPS) supports all Shuttle preparation and test activities from arrival at KSC through to launch. The LPS provides the sole direct real-time interface between Shuttle engineers, Orbiter vehicles and payloads, and associated Ground Support Equipment [He87].

The locus of control for the LPS is the Firing Room, an integrated network of computers, software, displays, controls, switches, data links and hardware interface devices. Firing Room computers are configured to perform independent LPS functions through application software loads. Shuttle engineers use computers configured as Consoles to remotely monitor and control specific vehicle and Ground Support systems. Each such application Console communicates with an associated Front-End Processor (FEP) computer that issues commands, polls sensors, and preprocesses sensor measurement data to detect significant changes and exceptional values. These computers are connected to data busses and telemetry channels that interface with Shuttles and Ground Support Equipment.

The LPS Operations team ensures that KSC's four independent Firing Rooms are available continuously, in appropriate error-free configurations, to support test requirements such as Launch Countdown or Orbiter Power-up sequences for the Shuttle fleet. A dedicated Console computer is configured for these Operations support functions in each Firing Room. This computer displays messages triggered by the LPS Operating system that signal anomalous events such as improper register values or expiring process timers. The Operations Console supports other conventional programs for monitoring and retrieving Firing Room status data as well.

OPERA (for Operations Analyst) consists of an integrated collection of expert systems that automates many critical LPS Operations support functions [Ad89b]. OPERA taps into the same data stream of error messages that the LPS sends to the Operations Console. OPERA's primary expert systems monitor the data stream for anomalies and assist LPS Operations users in isolating and managing faults by recommending troubleshooting, recovery and/or workaround procedures. In effect, OPERA ***retrofits*** the Operations Console with knowledge-based fault isolation capabilities. The system is implemented in KEE on Texas Instruments Lisp Machines.

GPC-X is a prototype expert system for isolating faults in the Shuttle vehicle's on-board computer systems, or GPCs. GPC-X monitors (simulated) PCM telemetry data to detect and isolate faults in communications between Shuttle GPC computers and their associated GPC-FEP computers in LPS Firing Rooms. The GPC-X prototype is implemented in CLIPS on a Sun Workstation.

## Coordinating Fault Diagnosis with HDC-Managers

One type of memory hardware fault in GPC computers manifests itself during switchovers of Launch Data Buses. These buses connect GPCs to GPC-FEPs until just prior to launch, when communications are transferred to telemetry links. Unfortunately, the data stream available to GPC-X does not provide any visibility into the occurrence of Launch Data Bus switchovers. Thus, GPC-X can propose, but not test certain fault hypotheses about GPC problems. However, switchover events are monitored by the LPS Operating System, which triggers messages that can be detected by OPERA.

Typical of the current generation of knowledge-based systems, OPERA and GPC-X were developed independently of one another, using different representation schemes, reasoning and control models, software and hardware platforms. More critically, neither system possesses internal capabilities for modeling or communicating with (remote) peer systems. The EXODUS prototype demonstrates the use of SOCIAL Agents to rectify these shortcomings (cf. Figure .6). The distributed application uses two Knowledge Gateway Agents to integrate OPERA and GPC-X. An HDC-Manager Agent mediates interactions between the OPERA and GPC-X Agents, coordinating their independent fault isolation activities obtain enhanced diagnostic results.

Specifically, GPC-X, at the appropriate point in its rule-based fault isolation activities, issues a request via its Gateway Agent to check for Launch Data Bus switchovers to the HDC-Manager. The request is triggered by adding a simple consequent clause of the form (GW-Return LDB-Switchover-Check) to the CLIPS rule that proposes the memory fault hypothesis. GW-Return is a custom

C function defined in the SOCIAL API interface for embedding CLIPS. When the rule fires, GW-Return interacts with the GPC-X CLIPS Gateway Agent, causing it to formulate a message to the HDC-Manager containing a Modify-Agenda request to add a MetaData Task object for the LDB-Switchover-Check service. The HDC-Manager's Command-Manager dispatches this request, causing the Task to be posted to the Task-Agenda.

Next, the HDC-Manager executes the Control-Cycle method, which results in the Task being processed via the Task Dispatcher. First, the Index knowledge base is searched for a server Agent for LDB-Switchover-Checks. The search identifies the OPERA Gateway Agent as a suitable server. Task data is then reformulated into a command message using the procedure specified by the Index Knowledge Base. The message is then passed to the OPERA Gateway Agent, whose in-filter method performs a search of the knowledge base used by OPERA to store and interpret LPS Operating System error messages. The objective is to locate error messages (represented as KEE units) indicative of LDB Switchover events. Search results are encoded within a Manager Bulletin-Board MetaData object, which the OPERA Gateway's out-filter method returns as a message containing an API command to post that object to the Manager's Bulletin-Board. In this prototype, the OPERA Gateway contains *all* of the task processing logic: OPERA itself is a passive participant that continues its monitoring and fault isolation activities without significant interruption.

The GPC-X CLIPS Gateway Agent queries the HDC-Manager to check the Bulletin-Board for a response to its LDB-Switchover-Check request and retrieves the results. The retrieved Bulletin-Board item is decoded and the answer is converted into a fact that is asserted into the GPC-X fact base. Finally, the Gateway activates the CLIPS rule engine to complete GPC fault diagnosis. Obviously, new rules have to be added to GPC-X to exploit the newly available hypothesis test data. However, all of the basic integration and coordination logic is supplied by the embedding GPC-X Gateway Agent or the HDC-Manager.

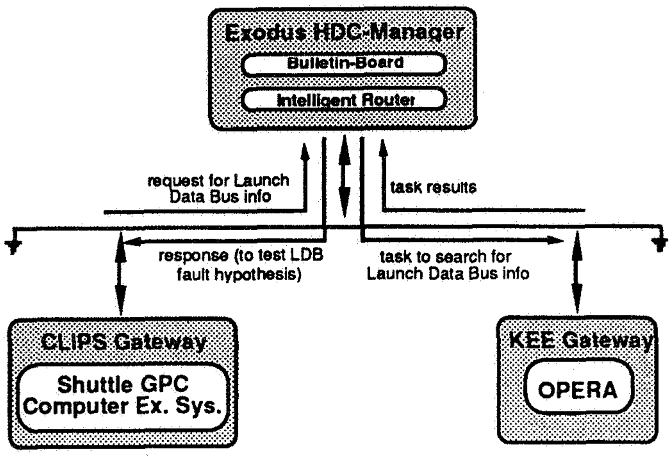

Exodus HDC-Manager
Bulletin-Board
Intelligent Router
request for Launch Data Bus info
task results
response (to test LDB fault hypothesis)
task to search for Launch Data Bus info
CLIPS Gateway
Shuttle GPC Computer Ex. Sys.
KEE Gateway
OPERA

Figure .6: Hierarchical Coordination in EXODUS

This EXODUS prototype illustrates non-intrusive system-level coordination of distributed applications that solve problems at the subsystem level of Shuttle Operations: neither OPERA nor GPC-X are capable of accomplishing the task of confirming or rejecting the GPC memory fault hypothesis individually. GPC-X generates fault candidates, but lacks sufficient resources to complete diagnosis, which requires *both* generate and test capabilities. OPERA automatically detect LPS error messages that are relevant to GPC-X's fault test requirements. However it lacks contextual knowledge about GPC computers - their architecture, behavior, fault modes and symptoms - to recognize the significance of such data. OPERA also lacks the capabilities to communicate its interpretations of LPS data back to GPC-X to complete diagnosis.

Together with the Gateway application Agents, the HDC-Manager provides the links required to combine and utilize the otherwise isolated or fragmented knowledge about Shuttle and Firing Room systems and their relationships to one another. The resulting coordination architecture is non-intrusive in that neither application was modified to include direct knowledge of the other system, its interfaces, knowledge model, or delivery platform. The HDC-Manager introduces an isolating layer of abstraction; application Agents need only know how to communicate with the HDC-Manager to request services and retrieve responses for their embedded applications.

The proposed design for the complete EXODUS system specifies an extended HDC model to integrate and coordinate all of KSC's operations support applications across areas of functional overlap. A SOCIAL HDC-Manager Agent will support architectural changes as EXODUS evolves through its lifecycle of initial deployment, maintenance, and enhancement. Initial EXODUS applications are loosely-coupled and will interact relatively infrequently. Consequently, the HDC-Manager's centralized control strategy does not entail serious performance penalties. As new applications are added and interaction traffic increases, bottlenecks will be addressed by reorganizing the EXODUS control architecture in terms of hierarchies of HDC-Manager Agents, much as growing human organizations evolve.

## RELATED WORK

The HDC-Manager generalizes and extends a hierarchical distributed control (HDC) model originally developed for NASA's Operations Analyst (OPERA) system [Ad89a]. The OPERA version of the control model was implemented using distributed blackboard objects [Ad89b]. OPERA requires all applications to be co-resident and to be implemented using KEE, and only supports a single Manager and subordinate group. The extended HDC model relaxes these restrictions using Meta-Courier, MetaData, API tools, SOCIAL Agents.

Most research in Distributed Artificial Intelligence (DAI) has focused on domains involving a single complex problem, such as data fusion. Control schemes have emphasized purely local coordination methods to achieve cooperation among intelligent systems [Bo88,Hu87]. For example, [Le83] employs a homogeneous collection of blackboards that interpret data from a (simulated) network of spatially distributed sensors to reconstruct vehicle positions and movements. Data and hypotheses are shared across adjacent sensor regions. Solutions emerge consensually, without global management. Decentralized control entails significant performance overhead from duplicated processing. As argued earlier, localized coordination strategies can be cumbersome and difficult to maintain for heterogeneous, evolving "multiple problem" DAI systems.

The MACE DAI tool [Ga86] incorporates manager agents for centralized routing of messages among agents, closely resembling the organizing role played by SOCIAL HDC-Managers. However, it is not clear that MACE managers can be configured in multi-level hierarchies. Moreover, MACE managers do not provide shared memory Bulletin-Boards. MACE and several other distributed system tools such as ABE [Ha88] and CRONUS [Sh86] insulate developers from low level distributed computing functions and support message sending among processes across computer networks. However, unlike SOCIAL, these tools do not implement generic distributed services in uniformly object-oriented layered modules that are accessible to developers for customizing. In addition, SOCIAL provides more extensive tools for integrating across languages and software development shells.

## FUTURE WORK

Future development will extend SOCIAL's library of Manager Agents. Alternative organizational models will explore alternative types of nonhierarchical cooperative coupling. For example, we are investigating control behaviors based on group-based tasking [Br89], as one approach to providing fault tolerance in distributed systems: groups can be used to define sets of application Agents that duplicate support for given services. A group Agent that could detect loss of an Agent configured to provide a service (e.g., due to dropped network links or host platform failures), could activate another member Agent to resume the service. Redundancy and recoverability are critical prerequisites for distributed systems in mission-critical space and military applications. We also plan to implement C versions of SOCIAL Manager and Gateway Agents that are currently Lisp-based.

## CONCLUSIONS

Development tools for distributed intelligent systems must be modular and non-intrusive to: (a) facilitate integration of existing, standalone systems "after the fact;" and (b) minimize lifecycle costs for maintaining, enhancing, and re-verifying systems. Tools for building distributed systems must be able to coordinate as well as integrate autonomous application elements. Coordination is necessary to manage large numbers of dynamic interaction pathways and to capture complex organizational relationships among application elements. SOCIAL provides a unified set of object-oriented tools that address all of these requirements. The HDC-Manager Agent realizes a hierarchical distributed control model that adopts a highly centralized approach to coordination. Developers use a high-level Application Programming Interface to access the HDC-Manager's coordination capabilities. The API conceals lower-level SOCIAL tools for transparent distributed communication, control, and data management.

SOCIAL has broad applicability for distributed intelligent systems that are being developed in space-related domains. Knowledge-based and conventional tools for managing and analyzing data need to be coupled to help space scientists explore and utilize NASA's growing stores of astronomical and environmental information. Linking short- and long-term scheduling and planning tools will improve decision support capabilities for complex space missions. EXODUS-like architectures can increase automation of operations support by coordinating autonomous tools across subsystems and functional areas (e.g., configuration, anomaly detection, diagnosis and correction). Example domains include payload and Shuttle processing, computer and communications networks, and vehicle or Space Station subsystems (e.g., power generation, power distribution, mission payloads, life support). Finally, flight and mission control centers can enhance automation and safety in directing launches, satellites, and space probes, by combining decision and operations support tools into fully unified, cooperating systems.

### Acknowledgments

SOCIAL was designed and implemented by the author, Bruce Cottman and Rick Wood. Development of SOCIAL has been sponsored by NASA Kennedy Space Center under contract NAS10-11606. Astrid Heard of the Advanced Projects Office at Kennedy Space Center has provided invaluable support and suggestions for the SOCIAL effort. Ms. Heard also initiated and directs the EXODUS project. MetaCourier was developed by Robert Paslay, Bruce Nilo, and Robert Silva, with funding support from the U.S. Army Signals Warfare Center under Contract DAAB10-87-C-0053.

## REFERENCES

[Ad90a] Adler, R.M. and Cottman, B.H. "EXODUS: Integrating Intelligent Systems for Launch Operations Support." *Proceedings, Space Operations, Applications, and Research Symposium (SOAR90).* Albuquerque, NM. June 26-28, 1990.

[Ad90b] Adler, R.M. and Cottman, B.H. "A Development Framework for AI Based Distributed Operations Support Systems." *Proceedings Fifth Conference on AI for Space Applications.* Huntsville, AL, May 21-23, 1990.

[Ad89a] Adler, R.M., Heard, A., and Hosken, R.B. "OPERA - An Expert Operations Analyst for A Distributed Computer Network." *Proceedings Annual AI Systems in Government Conference, Computer Society of the IEEE.* Washington, D.C., March 27-31, 1989.

[Ad89b] Adler, R.M. "A Distributed Blackboard Architecture for Integrating Loosely-Coupled Knowledge-Based Systems." *Intelligent Systems Review.* 1, 4, Summer, 1989, Association for Intelligent Systems Technology, E. Syracuse, NY.

[Br89] K. Birman et. al. *The ISIS System Manual V1.2.* Department of Computer Science, Cornell University, Ithaca, NY, June 1989.

[Bo88] Bond, A.H., and Gasser, L. (eds.) *Readings in Distributed Artificial Intelligence.* Morgan-Kaufmann, San Mateo, CA, 1988.

[Ga86] Gasser, L., Braganza, C., and Herman, N. ***MACE: A Flexible Testbed for Distributed AI Research.*** Distributed Artificial Intelligence Group, Computer Sci. Dept. USC, 9-Aug-1986.

[Ha88] Hayes-Roth, F., Erman, L.D., Fouse, S., Lark, J.S., and Davidson, J. (1988). "ABE: A Cooperative Operating System and Development Environment." in A.H. Bond and L. Gasser, (eds.) ***Readings in Distributed Artificial Intelligence.*** Morgan-Kaufmann, San Mateo, CA, 1988.

[Hu87] Huhns, M.N. (ed.) ***Distributed Artificial Intelligence.*** Morgan-Kaufmann, Los Altos, California, 1987.

[Le83] Lesser, V.R and Corkill, D.D. "The Distributed Vehicle Monitoring Testbed: A Tool for Investigating Distributed Problem Solving Networks." *AI Magazine.* Fall 1983 pp. 15-33.

[Sh86] Schantz, R., Thomas, R. and Bono, G. "The Architecture of the Cronus Distributed Operating System. ***Proceedings 6th International Conference on Distributed Computing Systems.*** May, 1986.

[Sy90] Symbiotics, Inc. ***Object-Oriented Heterogeneous Distributed Computing with MetaCourier.*** Technical Report, Cambridge, MA, March, 1990.

# MACHINE VISION BASED TELEOPERATION AID

William A. Hoff, Lance B. Gatrell, John R. Spofford
Martin Marietta Astronautics Group
Mail Stop 4372, P.O. Box 179
Denver, CO 80201
Phone: (303)977-4114

## ABSTRACT

When teleoperating a robot using video from a remote camera, it is difficult for the operator to gauge depth and orientation from a single view. In addition, there are situations where a camera mounted for viewing by the teleoperator during a teleoperation task may not be able to see the tool tip, or the viewing angle may not be intuitive (requiring extensive training to reduce the risk of incorrect or dangerous moves by the teleoperator). A machine vision based teleoperator aid is presented which uses the operator's camera view to compute an object's pose (position and orientation), and then overlays onto the operator's screen information on the object's current and desired positions. The operator can choose to display orientation and translation information as graphics and/or text. This aid provides easily assimilated depth and relative orientation information to the teleoperator. The camera may be mounted at any known orientation relative to the tool tip. A preliminary experiment with human operators was conducted and showed that task accuracies were significantly greater with than without this aid.

*Keywords*: Machine Vision, Teleoperation, Telerobotics, Pose Estimation.

## 1. INTRODUCTION

Telerobotics has the potential to greatly benefit many space applications, by reducing the great cost and hazards associated with manned flight operations. For example, space assembly, maintenance, and inspection tasks can potentially be done remotely using robots instead of using extra-vehicular activity (EVA). Teleoperation is an attractive method of control of such robots due to the availability and maturity of the technology. Unfortunately, using remote camera views degrades the operator's sense of perception as compared to actually having the operator physically on the scene. This paper describes how artificial intelligence (specifically, machine vision) can be used to implement a teleoperator aid that improves the operator's sense of perception.

### 1.1 The Problem of Perception in Teleoperation

In this paper, we are concerned with the class of teleoperation tasks that involves placing the end-effector of the robot in a certain pose (position and orientation) relative to some other object in the scene. This class of tasks includes most manipulation tasks, since generally an object must be grasped or manipulated in a specific manner, and so the accurate placement of the end-effector with respect to that object is a required precondition. In addition to manipulation requirements, the end-effector must be moved accurately around the workspace to avoid collisions. We are also concerned with the class of tasks in which the identity, geometry, and appearance of the object to be manipulated is well known in advance, but its location is only approximately known. Finally, we are concerned with tasks where the end-effector must be placed relative to the object with an accuracy that is tighter or more stringent than the initial *a priori* knowledge of the location of that object. This situation is common when the task and environment are fairly well specified in advance, but the exact locations of objects are uncertain due to manufacturing tolerances, measurement uncertainties, thermal effects, *etc.*

To provide an illustrative example, many proposed space robotics tasks, such as engaging bolts and mating connectors, are estimated to require positional accuracies of as tight as ±0.125" and ±1° [GSFC87]. On the other hand, the absolute positioning accuracy of the Flight Telerobotic

Servicer (FTS) robot is required to be only ±1", ±3° (this refers to the position of the end effector relative to the robot's stabilizer arm attachment point). In addition to this, there are uncertainties in the robot's docking or attachment mechanism, and uncertainties in the position of the object in the workspace. These can potentially add several inches and degrees to the total end effector-to-object uncertainty.

The net result is that the motion of the robot cannot be pre-programmed in advance, but that some sort of sensor feedback must be used to correct for positioning errors. In the case of teleoperation, the sensor feedback to the operator usually consists of remote camera video and force reflection. Force reflection is only useful when the end effector has already made contact with the object, and may be used in some cases to correct for very small positioning errors. However, to move the end effector from a potentially distant position to close proximity of the object while avoiding obstacles (at which point a "guarded" move can be performed), visual feedback is necessary.

For monoscopic vision, using a single wrist mounted camera (or possibly a head camera in addition to a wrist camera), the operator may have difficulty in perceiving the three dimensional relative position and orientation of objects in the scene. Absolute distances and orientations are even more difficult to judge. In addition to these problems, in some cases the cameras are not able to get a good view of critical locations such as the tool tip, due to occlusions. Extensive operator training can alleviate these problems, but this training must be very specific to a particular task and workspace. Any changes to either the object or the background can mislead the operator and cause errors.

## 1.2 Approaches to Solving the Perception Problem

There have been a large number of studies performed over the years on the effects of the characteristics of video displays on the ability of the operator to perform manipulative tasks. A single video display usually does not provide good task perspective. Generally, studies have found that stereoscopic vision is superior to monoscopic for typical manipulation tasks because it provides a better sense of depth perception [Pepp83]. Two separate, preferably orthogonal, camera views can also be used to give more perspective. However, this approach requires the operator to look at two displays, and worksite camera locations may be limited. Also, two camera views of equal resolutions can require twice the communications bandwidth on a single camera, which is an important consideration in remote space operations.

Both perspective and stereoscopic visual cues have been shown to improve manual tracking and manipulation ability. A perspective cue that provides information not directly indicated by the task image appeared to improve performance the most in simulation studies [Kim87a],[Kim87b]. A 2D display of a 3D tracking task tends to increase errors of translation into the image and rotations about axes in the image plane [Mas89]. Graphics superimposed on a video view has been previously shown to assist operators in grasping objects with a manipulator [Kim89]. That work used an *a priori* modelled location of the camera and manipulator base to draw a graphic aid based on the current manipulator pose.

## 1.3 The Machine Vision Approach

In this paper, we propose an alternative solution to the perception problem that is based on the use of artificial intelligence. Specifically, we use machine vision to automatically recognize and locate the object in the camera views, and then display the pose of the object as an overlay on the operator's live video display. The operator can thus teleoperate the robot guided by the computed pose information, instead of or in addition to the video image. We have implemented this system and have measured its benefit to teleoperators in a preliminary experiment.

The remainder of this paper is organized as follows. Section 2 describes previous approaches to teleoperator aids and discusses the advantages of our approach over existing techniques. Section 3 describes our teleoperator aid that we have implemented and evaluated at Martin Marietta. Section 4 summarizes the machine vision technology that is used to derive the object's pose. Section 5 summarizes the remainder of our laboratory facilities, including the robot manipulators, controllers, and operator workstation. Section 6 describes the experiment that was conducted to measure the effect of the teleoperator aid on operator performance. Section 7 gives conclusions.

## 2. PREVIOUS TELEOPERATOR AIDS

Without a teleoperator aid, the operator must rely on his visual memory of where the end effector should be in the images to gauge his accuracy. In our task, described in Section 6, there were no stadiametric background marks with which to estimate the end effector position. Without a teleoperator aid, we found that the operator was accurate to approximately five millimeters and about two degrees rotation about any one axis. If the operator knew what his pose errors were, he could reduce them to the accuracy limits of the manipulator and the sensor. A sensor which can inform the operator of his pose errors gives the operator the ability to reduce those errors. With such a sensor, one would expect the operator's performance to be more accurate and repeatable since the operator has more accurate feedback information. Furthermore, one would expect that an inexperienced operator using the sensor could achieve better accuracy and repeatability than an experienced operator working from memory alone — at a fraction of the cost for training.

The most basic example of teleoperator aids is the printed transparent overlays used by astronauts operating the RMS arm to grapple an RMS target, and as a backup mode for final rendezvous of the Space Shuttle orbiter with other spacecraft. Each object that is to be used with the transparency aid has its own set of transparencies — typically at least one for full camera zoom, and one for no camera zoom. Tick marks are placed on the transparency so that the operator can easily determine the approximate depth to the object by how many tick marks are filled in the image by the object. The roll of non-symmetric objects can also be determined if radially converging lines are placed on the object. For the case of the RMS target overlays, the RMS is positioned correctly at an RMS target when the target matches the pattern printed on the overlay.

In related work, Bejczy [Bejc80] has described a teleoperator aid that uses four proximity sensors on the end effector of the Remote Manipulator System (RMS) to compute pitch, yaw, and depth information and display it to the operator. The advantages of our system over this work is that we use video sensors, thus achieving a much greater range of operation (depths up to 72 cm versus 15 cm, and yaw angles of [-25°..40°] versus ±15°), we compute all six degrees of freedom instead of only three, and we have a display that is integrated with the video instead of separate.

Another related work of interest was reported by Hartley and Pulliam [Hart88]. Seven display aids for use by pilots of Remotely Piloted Space Vehicles (RPSV) were presented. These display aids were developed during the course of simulations of RPSV tasks at Martin Marietta's Space Operations Simulation laboratory, where a moving base carriage robot was used to simulate RPSV's. Three of the remote pilot aids were reticles that were overlaid on the operator's camera view that graphically gave the operator feedback on the vehicle's pose and trajectory relative to the target. The remaining aids were basically displayed patterns, such as the RMS docking target, which would match the actual target when the RPSV achieved the goal pose — a dynamic version of the transparencies used by astronauts controlling the RMS. Data for these pose and trajectory aids was obtained by reading the moving base carriage pose information. In a real task, this information would have to be sensed by sensors.

One difference in the work of Bejczy compared to that of Hartley is that Bejczy's sensor display was on specialized boxes, whereas Hartley's displays were overlaid on the principal camera view screen. The advantage of on screen display is that the operator does not have to change his view to see the sensor output. Graphic display devices that allow display of live video and overlaid graphics simultaneously are typically higher resolution devices than standard NTSC monitors, and consequently, alignment aids can be more precisely overlaid in the image, further helping to reduce teleoperator errors.

The principal element of our approach is the machine vision based sensor that processes the operator's wrist camera view to compute the transformation (pose errors) between the current end effector position and the goal position. The pose errors are then overlaid on the wrist camera portion of the operator's high resolution monitor as graphics and text.

Some of the advantages of this system are as follows: No additional sensor is required on the manipulator since it uses the operator's video images. All six degrees of freedom of the pose are presented to the operator, with the data display integrated into the operator's screen so that the operator does not have to change his view to see the data. The displays are dynamic in that they are computer generated for the appropriate object as indicated by the operator — without the need for carrying physical transparencies as currently used by the RMS operators. Finally, the computer is able to produce data that is more accurate than the human operator could produce from the same camera view.

## 3. THE MACHINE VISION-BASED TELEOPERATOR AID

### 3.1 Coordinate Systems

Figure 3.1-1 illustrates the primary coordinate systems, or frames, that are involved in the computations of our teleoperator aid. In this Section, we use the naming and notation conventions of Craig [Crai89]. The station frame, $\{S\}$, is fixed in the world and is attached to the object that is to be manipulated. The tool frame, $\{T\}$, is attached to the tool or end effector of the robot arm. The desired position of the tool is given by the goal frame, $\{G\}$. Specifically, when the tool is in the desired position, the tool frame coincides with $\{G\}$. The camera frame, $\{C\}$, is attached to a camera that is rigidly mounted on the wrist of the robot arm. When the robot arm is not in the desired goal position, let the tool frame be represented by $\{T'\}$ and the corresponding camera frame by $\{C'\}$, as shown in Figure 3.1-1. The information needed by the teleoperator is how to transform the current tool frame $\{T'\}$ into the desired tool frame $\{T\}$.

We use the following notation conventions: ${}^{A}_{B}T$ represents a homogeneous transformation from frame B to frame A, expressed as a 4x4 matrix. ${}^{A}_{B}R$ represents the rotation portion of such a transformation (a 3x3 matrix), and ${}^{A}P_{BORG}$ represents the translational portion (a 3x1 matrix), which is also the location of the origin of the $\{B\}$ frame in the coordinate system of $\{A\}$.

The machine vision system can provide the location of the object with respect to the camera, *i.e.*, ${}^{C}_{S}T$ and ${}^{C'}_{S}T$. To set up our system, we initially moved the robot arm to the goal position and recorded ${}^{C}_{S}T$. During operation, we then moved the arm away and tried to determine the transformation necessary to transform the current tool frame into the goal frame ${}^{T}_{T'}T$, based on an observation of ${}^{C'}_{S}T$. If we knew the transformation from the tool frame to the camera frame, ${}^{C}_{T}T$, we could find the desired transformation as follows:

$${}^{T}_{T'}T = {}^{T}_{C}T \; {}^{C}_{S}T \; {}^{S}_{C'}T \; {}^{C'}_{T'}T$$

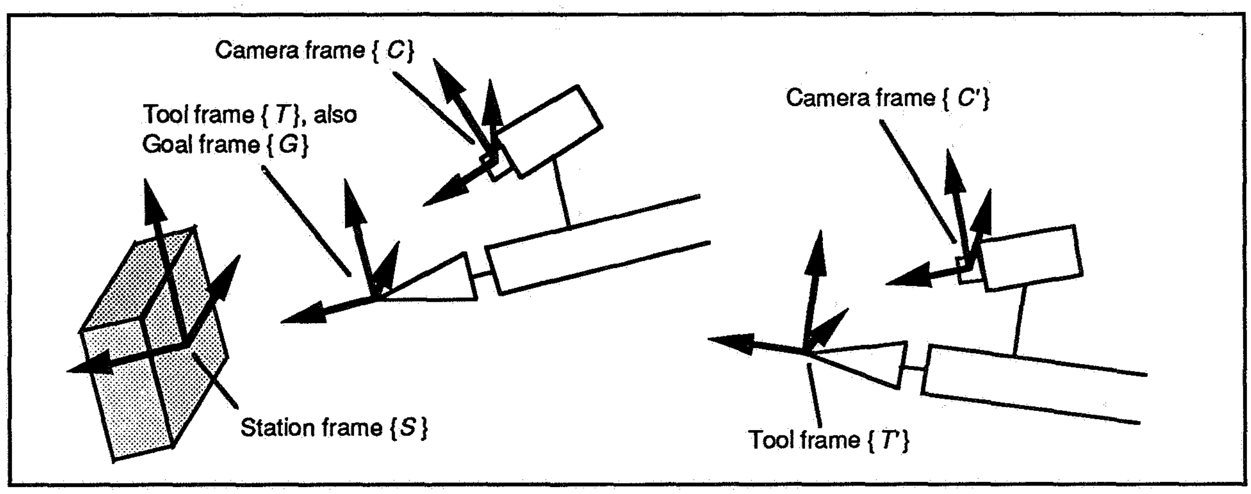
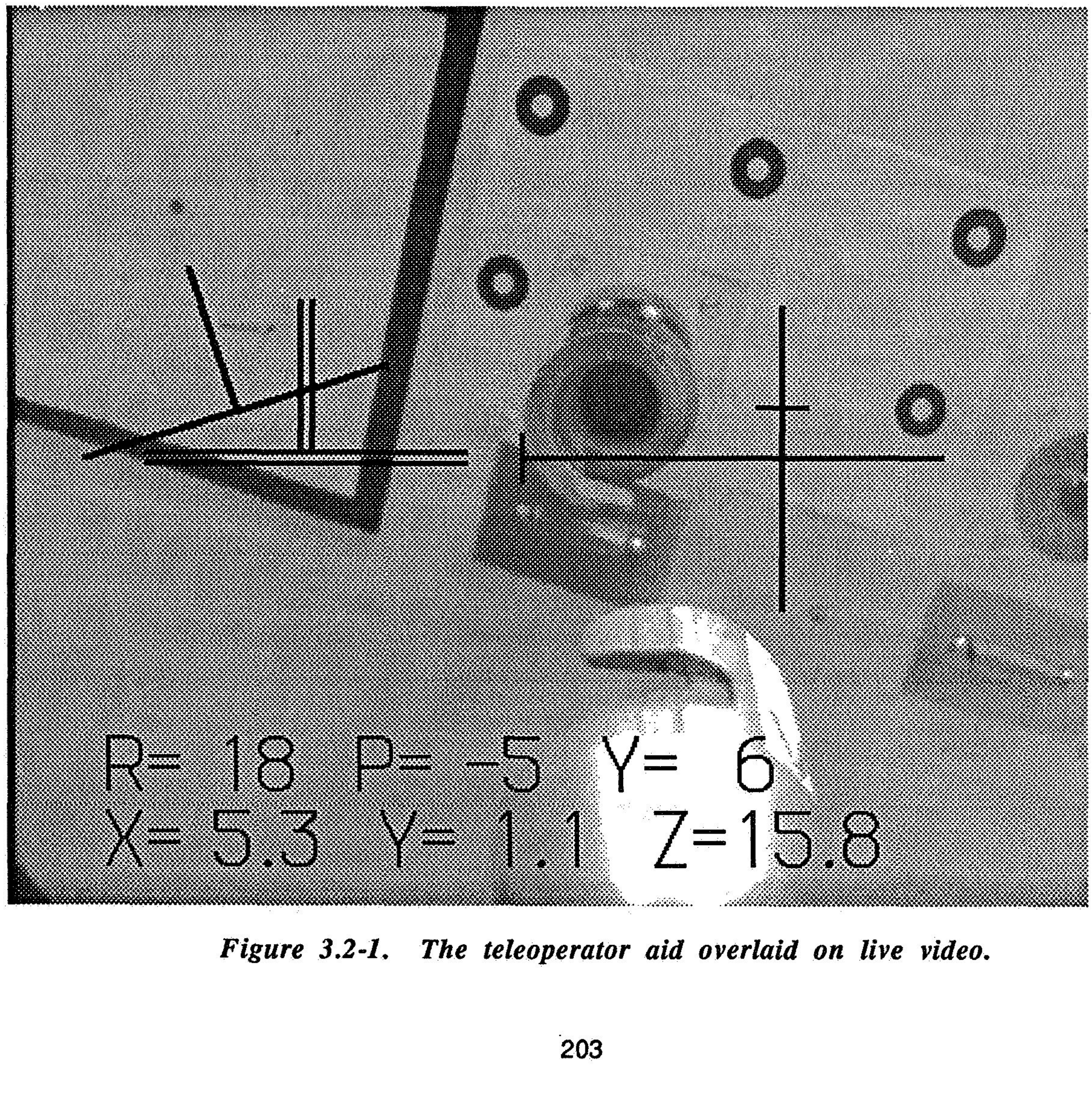


*Figure 3.1-1. The primary frames, or coordinate systems, involved in the machine vision based teleoperation aid.*

However, in the teleoperator aid implementation described in this paper, we did not know ${}^{C}_{T}T$, although in principle this can be measured. Nevertheless, we were still able to compute useful information about the transformation from $\{T'\}$ to $\{T\}$, because of a special relationship between our station frame $\{S\}$ and our goal frame $\{G\}$. The special relationship is that the axes of these two frames were parallel; *i.e.*, there was only a translation between them. The effect of this was that we were able to compute ${}^{T}_{T'}R$, but not ${}^{T}P_{T'ORG}$. Instead of computing the true translation between the current tool frame $\{T'\}$ and the final tool frame $\{T\}$, the translation that we computed had an additional component due to rotation. However, when there was no rotation, the translation that we computed was correct. This was actually not confusing when we used the teleoperator aid. If we first rotated the end effector to zero the orientation errors, then we could simply translate the end effector according to the displayed pose. Appendix 1 provides a detailed explanation of this. In future work, we plan to measure the camera-to-tool transform so that we can directly compute and display the correct $\{T'\}$ to $\{T\}$ transform.

## 3.2 Display of Coordinates

Our teleoperation aid displayed the computed pose error as both text and graphics overlaid on the live video. Figure 3.2-1 shows a screen photograph of the text and graphics overlaid on a video image of our truss connector and panel. The lower portion of the screen is occupied by the text. Pose error is reported as the translation in centimeters along the X, Y, and Z axes; and the rotation in degrees about the X, Y, and Z axes (also called pitch, yaw, and roll).



```
R= 18 P= -5 Y= 6
X= 5.3 Y= 1.1 Z=15.8
```

***Figure 3.2-1. The teleoperator aid overlaid on live video.***

Orientation is also displayed using a reticle, an upside-down "T", in the left center of the screen. Yaw errors are indicated by horizontal translations of the "T", and pitch errors by vertical translations. Roll errors are indicated by a rotation of the "T". By looking at the orientation reticle in Figure 3.2-1, the operator can tell that to get to the goal orientation, he must roll the arm to the right, pitch it down, and yaw to the left. When all errors are zero, the "T" fits exactly inside a hollow fixed reference "T", as shown in Figure 3.2-2.

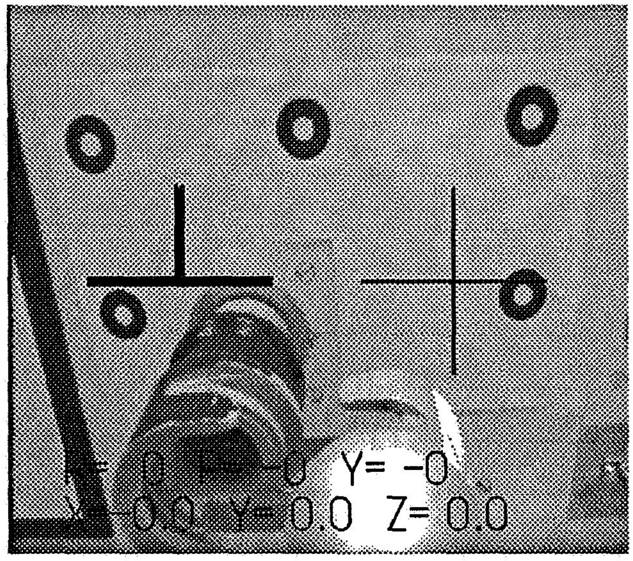


*Figure 3.2-2. The display when pose error is zero.*

Translation is displayed using a pair of error bars in the right center of the screen, one horizontal and one vertical. Each bar is terminated by a crosspiece at right angles to the bar. Horizontal errors are indicated by a lengthening of the horizontal bar, and vertical errors are indicated by a lengthening of the vertical bar. When the depth error changed sign, the fixed reference crosshairs changed color.

## 4. MACHINE VISION SYSTEM

The goal of our machine vision system was to identify and estimate the pose of an object of interest in the scene. Although significant progress has been made in the field of machine vision, no system exists at present which can identify large numbers of different objects against multiple backgrounds at video update rates. One alternative is to place visual targets, which can be recognized at video rates, on the objects. Visual targets which have been used to simplify the object recognition process are summarized by Gatrell, *et al.* [Gatr91].

### 4.1 Image Features

The Concentric Contrasting Circle (CCC) image feature, developed at Martin Marietta and reported in [Gatr91, Skla90], is used in this work for the feature that the machine vision system is looking for. A CCC is formed by placing a black ring on a white background, and is found by comparing the centroids of black regions to the centroids of white regions — those black and white centroids which are equal are CCCs. This image feature is invariant to changes in translation, scale, and roll, and is only slightly affected by changes in pitch and yaw, and can be extracted from the image rapidly with low cost image processing hardware. The centroid of a circular shape is the most precisely locatable image feature [Bose90].

### 4.2 Object and Target Model

Four CCC's are placed in a flat rectangular pattern on the object to be recognized by our vision system. A fifth CCC is placed on a side of the rectangle to remove the roll ambiguity. This five point target is also described in more detail in [Skla90]. Our basic object recognition process has been reduced to the simple steps of finding five Concentric Contrasting Circles which form a five point Target. We have found this to be very robust and fast. In designing the five point target for a particular object, care must be taken to ensure that all five CCCs will be visible from the expected viewing positions. The target for the truss connector measured 16.5 cm x 6.3 cm; the diameter of the CCCs was 2 cm.

### 4.3 Pose Estimation

After features of an object have been extracted from an image and their correspondence between image features and object features has been established, the pose of the object relative to the camera, or ${}^{S}_{C}T$, can be computed by many techniques, such as [Chan89, Kris90]. We currently use the simple and fast Hung-Yeh-Harwood pose estimation method [Hung85]. The inputs to the pose algorithm are the centers of the four corner CCCs, the target model, and a camera model. The pose algorithm essentially finds the transformation which yields the best agreement between the measured image features and their predicted locations based on the target and camera models.

### 4.4 Camera Calibration

Accurate pose estimation requires an accurate camera model, including the focal length $f$ used above. Most pose estimation techniques, including the Hung-Yeh-Harwood method, assume a pin hole camera model with no lens distortion. Real cameras are not pin hole cameras, and real lenses have noticeable distortion--especially radial lens distortion. The characteristics of a lens, camera, and digitizer configuration are determined by camera calibration. We use the Tsai camera calibration technique [Tsai87] to compute the focal length and radial lens distortion, as discussed in [Skla90]. A picture of the block used to calibrate our camera is shown in Figure 4.4-1. Once again, the image features are the centers of the circles.

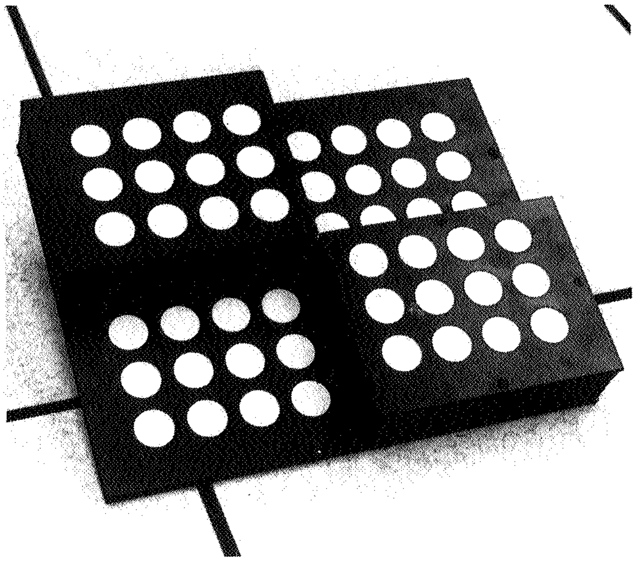


*Figure 4.4-1. Block used to calibrate the camera.*

During use of our operator aid, the precomputed radial distortion coefficients are used to "undistort" image feature locations on the image plane so that they fit a pin hole camera model. The undistorted image locations are given to the pose estimation routine described above. It should be noted that the Tsai calibration technique also computes the pose of targets relative to the camera, but is not used for estimating the pose of our five point target due to an insufficient number of points for this technique.

### 4.5 Equipment Used

In our current configuration the wrist camera was a high resolution, black and white, Pulnix TM-840 camera with an 8 mm wide angle auto-iris lens. The shoulder camera was a Panasonic AG-450 S-VHS color camera. Our digitizer and image processing board was an Androx ICS-400XM9. This DSP chip-based board computed the histogram and thresholded the image. The thresholded image was processed on the host machine, a Solbourne 4/501 running at 20 MIPS (Sun SPARC compatible), that computed the connected regions, extracted the CCCs, found the five point target, and computed the pose. The graphics overlays were drawn by the ICS-400XM9 board.

### 4.6 Vision System Capabilities

The vision system can be in one of three states: 1) not tracking any object, in which case the video is passed through to the operator's monitor, 2) acquiring the object — the last location of the object in the image is unknown, and therefore, the entire image is searched for the object, and 3) tracking the object — the object was found in the last view that was processed, and therefore, each of the target features is searched in a small area centered at the last known location. The acquire step takes about 0.8 seconds once a view with the target visible is acquired. The delay between when a target is visible to the camera and the acquire has finished could take up to 1.6 seconds (up to 0.8 seconds to finish searching the last occluded target view, and 0.8 seconds to process the visible target view). The track step takes about 0.1 to 0.2 seconds per cycle (5-10 Hz.), depending on the size of the CCC target features in the image. As each target feature increases in size, the portion of the image that has to be processed also increases, thus reducing the throughput rate.

A button-input panel on the Solbourne workstation screen allows the user to partially customize the operator aid display and to select from possible options. Aid display features which the operator can choose to have displayed are: 1) colored cross hairs on the CCC centroids, 2) wireframe overlay of the model, 3) orientation reticle, 4) orientation text, 5) translation reticle, and 6) translation text. The operator has the option of setting the current camera to object pose as the goal pose, and the option of reading in a goal pose from a file or writing the current goal pose out to a file, and the option of setting the goal pose to be the object's coordinate frame. Furthermore, the operator can adjust the image contrast with two sliding bars. At present, these capabilities are only selectable on the vision processor workstation screen; our goal is to integrate all of theses features on the high resolution operator console.

With our 8 mm wide angle lens, the target is found at depth ranges of 18 cm to 72 cm. The limiting factor in how close the camera can be to the target is the width of the target — at closer distances the target does not fit in the image. The target can be pitched from -32° to 77°, and can be yawed from -25° to 40°. The pitch and yaw limits are not symmetric because the truss connector extends in front of the target, and as the target pitches and yaws, the connector may occlude some of the CCCs.

# 5. LABORATORY FACILITIES

This section provides an overview of the facilities in the Combined Robotic Technologies Laboratory (CRTL) where this study was conducted. This laboratory contains a testbed for research in the areas of robotics, teleoperation, applied controls, computer vision, motion planning, situation assessment, and human factors. The laboratory processing architecture allows autonomous and manual control of robotic systems. A more complete description of the laboratory testbed can be found in [Mor90].

## 5.1 Processing Architecture

Figure 5.1-1 shows the functional architecture for the laboratory testbed. The architecture is based on the NASREM [Alb89] hierarchical model and equivalent NASREM levels are indicated. We have partitioned the system horizontally into three categories; Control and Automation, Situation Assessment and Operator Interfaces. Control and Automation subsystems provide control functions to testbed components or process sensor data used during the control function. Operator Interface subsystems include units which accept operator input and generate commands for Control and Automation units. It also includes units which display information to the operator or record information for analysis. The Situation Assessment subsystem provides the reasoning and model update functions for Control and Automation units which are not used during teleoperation. The study described in this paper used the control functions and operator interfaces at the Prim and Servo levels. A developmental version of the Vision sensing system at the Prim level was used for the operator aid function.

## 5.2 Robot Manipulators

The CRTL contains two 6 DOF Cincinnati Milacron T$^3$-726 robot manipulators and one T$^3$-746 manipulator. All three robots are commercial robots whose control systems have been replaced with custom controllers. They are configured as a dual-arm system with a dynamic task positioning system. Six-axis force/torque sensors are mounted between each manipulator's wrist and end effector. Video cameras are also mounted to the wrist of each manipulator. The servocontrollers for the manipulators receive Cartesian position commands from either the inter-arm coordinator or hand controllers. The commanded position is modified by an impedance control loop, also known as active compliance, based on sensed forces and torques. One of the T$^3$-726 manipulators was used with a fixed task panel in this study.

## 5.3 Operator Console

The CRTL has two generations of teleoperation control stations—a three-bay console and a two-bay console. The three-bay console has a center stereo display and two side displays with touch-screen overlays. The center stereo monitor can display live video from a stereo pair of cameras, overlaid with stereo graphics. The operator displays are generated by Silicon Graphics IRIS workstations with video underlays from RGB Spectrum video windowing systems. The resulting displays have 1024 by 1280 pixel resolution and can display video in a variety of resolutions, including a stereo aspect ratio.

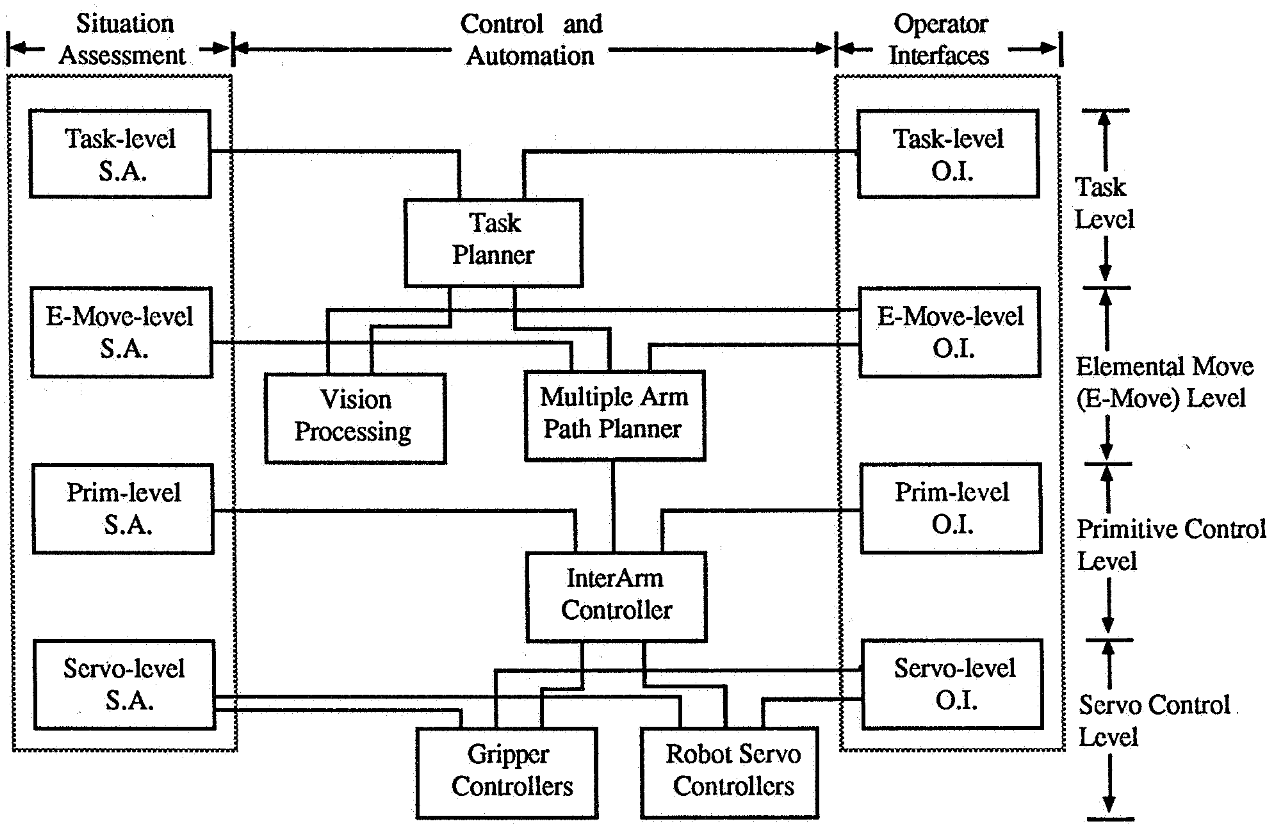


***Figure 5.1-1. Functional Architecture for the Tele-Autonomous Testbed***

The CRTL control stations are configured to use a wide variety of 6 DOF hand controllers. The Martin Marietta Compact ball was used in this study as the Servo level input device. The Compact ball hand controller is a Cartesian mechanism, with three translational joints and three rotational joints that position and orient the grip, respectively. The mechanical separation of these motions provides natural decoupling between Cartesian axes. A manipulator activation switch on each hand controller enabled manipulator motion when depressed and provided position mode indexing capability.

## 6. EXPERIMENT

We performed an experiment with human subjects to determine the effectiveness of the operator aid. This was a pilot study, intended to identify areas for more comprehensive future experimentation.

### 6.1 Experimental Design

Due to time limitations, a single experimental parameter and two subjects were used in this experiment. No comparisons between subjects or conclusions about general populations can be made with this few subjects. However, the experiment was designed so that each subject served as their own control, enabling conclusions about the effectiveness of the aid for each individual subject. The experimental parameter of interest was the presence of the operator aid on the video display, with two levels: present and not. Eight random starting locations were used to form a sample of the subject's performance, thus each subject performed a total of 16 runs. Both subjects were right-handed male engineers with previous experience operating the manipulator and hand controller. Neither had previous experience with the operator aid.

## 6.2 Task Description

The task selected for this experiment was an unobstructed free-space motion from unknown initial locations to a specified goal location. The task was defined as the final free-space move in a truss node assembly sequence, moving the manipulator-held connector half to a specified position relative to the mating connector half, with no orientation error. Motion in all 6 DOF was required. When the operator aid was displayed, the goal location was indicated by zero position and rotation error. Without the operator aid, only image visual cues were available.

The operator's feedback was from two camera views: the wrist camera was attached to the end effector above and behind it, tilted down at an angle of 18 degrees, while the second stationary camera was placed about 1.2 meters behind and 1.1 meters to the right of the panel mounted truss connector half.

Figure 6.2-1 shows the task panel, the two connector halves, and the camera mounted on the manipulator wrist. The subjects were presented with two video views of the task, one from the fixed color camera and one from the monochrome wrist camera. The manipulator was controlled in tool (end effector) frame, with the hand controller reference frame aligned with the connector seen in the wrist camera view. The tool command frame was not aligned with the wrist camera view, which was pitched down. The hand controller commands were filtered with a cutoff frequency of 4 Hz. Manipulator motion was commanded in position mode, with scaling factors of 0.6 and 0.2 for translations and rotations respectively.

Figure 6.2-2 shows a display similar to that presented to the subjects; the subjects did not see the graphical buttons around the screen periphery. The fixed camera view was full-screen (1:2 pixel mapping) and the wrist camera was quarter-screen (1:1 mapping), located in the upper right corner. Neither the camera views nor the displays could be modified by the operators during testing. The distance and orientation change from each of the eight random starting locations to the goal location are listed in Table 6.2-1.

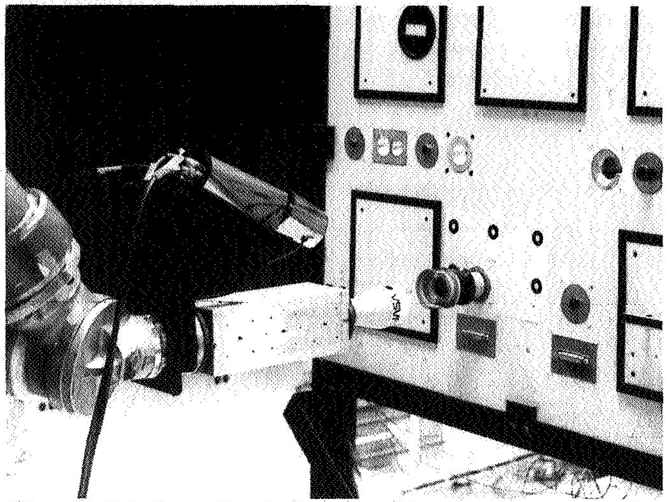


***Figure 6.2-1 Task Setup with Connectors and Wrist Camera***

***Table 6.2-1 Displacement and Orientation from Start Locations to Goal***

| Start Location | Displacement (cm) | Orientation (deg) |
|---|---|---|
| 1 | 19.50 | 4.46 |
| 2 | 20.82 | 20.41 |
| 3 | 31.02 | 20.45 |
| 4 | 26.90 | 17.72 |
| 5 | 22.61 | 15.68 |
| 6 | 16.34 | 15.08 |
| 7 | 21.34 | 20.81 |
| 8 | 15.55 | 15.84 |

## 6.3 Test Procedure

Each subject performed the experiment as a training session followed by the 16 data collection trials. The whole activity took about 90 minutes per subject. Table 6.3-1 lists the experimental conditions, which were randomly chosen for each subject.

Training was provided to the subjects to familiarize them with the task and display. The subjects were shown the goal position and a subset of the trial starting locations. The subjects were then allowed to operate the manipulator with the hand controller for a self-paced session. They could turn the operator aid on and off between and during training runs.

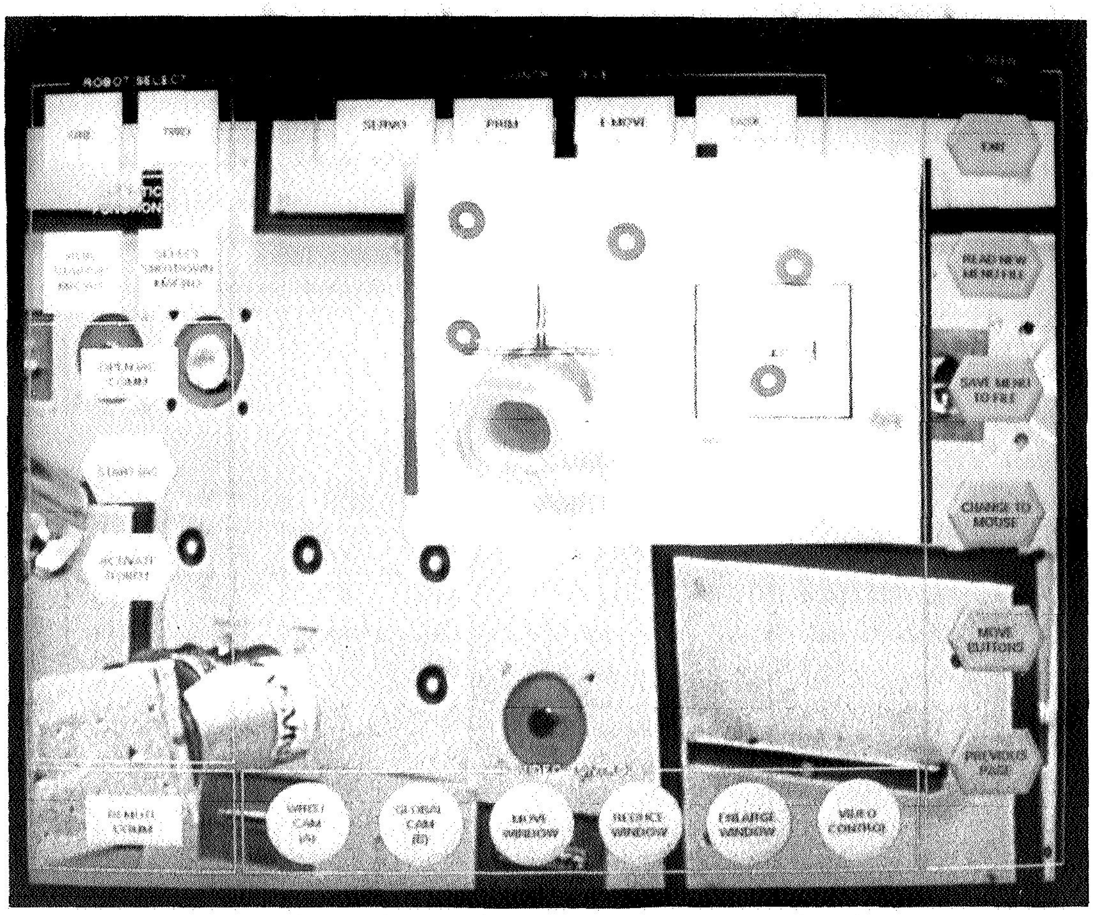


*Figure 6.2-2. Screen Display of Fixed and Wrist Cameras*

The subjects were able to practice from arbitrary start locations, but not the locations used in the actual experiment. The training sessions lasted until the subjects were comfortable with the system, typically about a half hour. Accuracy was specified as being more important than time in the completion of the task, with magnitudes of 0.2 cm and 1 deg being acceptable.

The trial runs were conducted by using the autonomous control system to move the manipulator to the appropriate start location, and then switching control to the subject in teleoperation mode. The subject was timed using two stopwatches from the start signal until the subject's indication of task completion. The position and orientation errors were calculated from the average of the manipulator's final pose and the vision system's final pose. These generally agreed to within 0.02 cm and 0.2 deg. The subject was not given an indication of performance at the end of each trial, except for the operator aid display when used.

## 6.4 Analysis Method

The method used here to test for the effect of the operator aid is to test for the possibility that the actual mean for the subject could be the same with and without the aid. This test can be done by comparing the ranges of each performance measure with and without the aid, at a specified See [Hic73] for a more detailed discussion of experimental design and analysis methods.

In the following analysis, the variable $X$ refers to any of the three performance measures: the task time, the position error, or the orientation error. The assumption of a normal distribution for the performance measures was made. Since the sample size is moderately small, a two-tail confidence level of 99% was chosen:

$$P\left[\bar{X}-C<\mu<\bar{X}+C\right] = 0.99$$

This states that the actual mean value $\mu$ for each subject is 99% likely to be within $C$ of the sample mean $\bar{X}$, where

$$c = t_{0.995} \frac{s_X}{\sqrt{n}}$$

and $\bar{X}$ and $s_X$ are the mean and sample deviation of the $n$ trials by each subject for each aid condition. $t_{0.995}$ is the value of the *t* distribution with a 99% two-tail confidence level. For a sample with $n = 8$ there are 7 statistical degrees of freedom, and $t_{0.995} = 3.499$.

***Table 6.3-1 Experimental Conditions***

| subject: | 1 | | 2 | |
|---|---|---|---|---|
| trial | start loc | aid | start loc | aid |
| 1 | 6 | YES | 1 | YES |
| 2 | 4 | YES | 2 | YES |
| 3 | 5 | no | 7 | no |
| 4 | 8 | YES | 3 | no |
| 5 | 6 | no | 2 | no |
| 6 | 1 | YES | 1 | no |
| 7 | 2 | YES | 6 | YES |
| 8 | 3 | YES | 6 | no |
| 9 | 7 | no | 4 | no |
| 10 | 7 | YES | 5 | no |
| 11 | 2 | no | 3 | YES |
| 12 | 1 | no | 8 | no |
| 13 | 4 | no | 7 | YES |
| 14 | 8 | no | 4 | YES |
| 15 | 5 | YES | 8 | YES |
| 16 | 3 | no | 5 | YES |

Treating the two aid conditions as separate population samples for each subject, a test can be made to see if they could be samples from the same population. If the 99% confidence ranges of the two samples do not overlap, then we can be confident that the actual means for both populations cannot be the same, indicating that the effect of the operator aid on the performance measure is statistically significant for the specific subject.

## 6.5 Results

Table 6.5-1 lists the sample mean, sample deviation, and high and low confidence limits of the three performance measures for the two subjects.

The following series of figures shows the performance of the two subjects. Each figure shows the sample mean and high and low 99% confidence values for a performance measure with and without the operator aid.

***Table 6.5-1 Experimental Data Summary***

| | | Subject | 1 | Subject | 2 |
|---|---|---|---|---|---|
| | | w/ aid | w/o aid | w/ aid | w/o aid |
| TASK | $\bar{X}$ | 47.1 | 86.1 | 82.4 | 63.1 |
| TIME | $s_X$ | 13.4 | 31.4 | 20.8 | 32.8 |
| (sec) | $\bar{X}+c$ | 63.6 | 124.9 | 108.2 | 103.7 |
| | $\bar{X}-c$ | 30.5 | 47.2 | 56.7 | 22.5 |
| POS | $\bar{X}$ | 0.20 | 0.68 | 0.16 | 1.18 |
| ERR | $s_X$ | 0.08 | 0.28 | 0.04 | 0.56 |
| (cm) | $\bar{X}+c$ | 0.30 | 1.03 | 0.21 | 1.88 |
| | $\bar{X}-c$ | 0.10 | 0.33 | 0.11 | 0.48 |
| ANG | $\bar{X}$ | 0.34 | 1.80 | 0.70 | 3.12 |
| ERR | $s_X$ | 0.24 | 0.74 | 0.37 | 0.76 |
| (deg) | $\bar{X}+c$ | 0.64 | 2.72 | 1.17 | 4.06 |
| | $\bar{X}-c$ | 0.05 | 0.88 | 0.24 | 2.18 |

Figures 6.5-1 and 6.5-2 show the task time measure. The average for subject 1 was faster with the aid than without, while the average for subject 2 was somewhat slower. This result will be discussed later. For both subjects, the confidence ranges overlap and no conclusion can be drawn regarding the benefit of the aid. In both cases, the variability of the task time decreased with the aid.

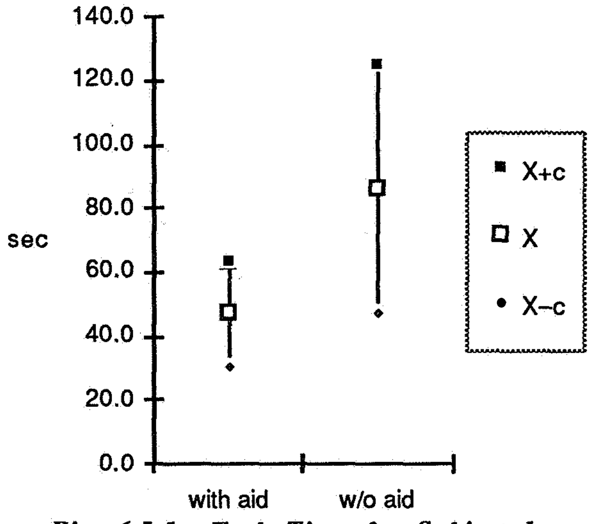


***Fig. 6.5-1 Task Time for Subject 1***

Figures 6.5-3 and 6.5-4 show the position error measure. Both subjects exhibited less error with the aid, and less variation. The confidences do not overlap, indicating a significant difference related to the presence of the operator aid.

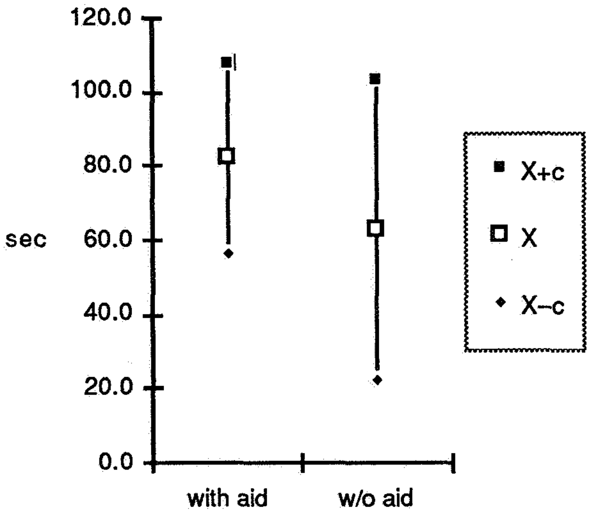


*Fig. 6.5-2 Task Time for Subject 2*

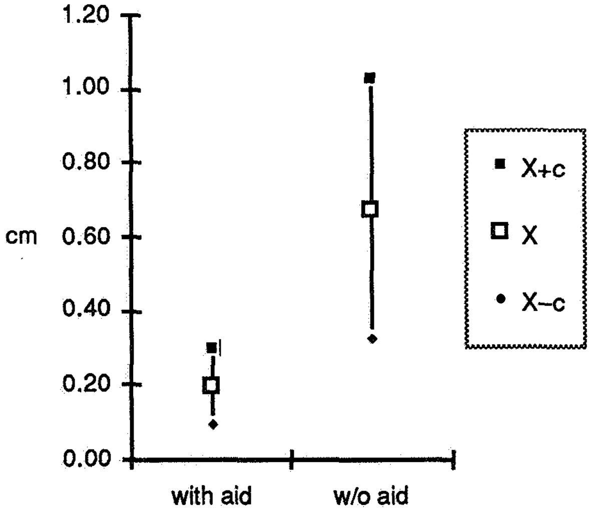


*Fig. 6.5-3 Final Position Error for Subject 1*

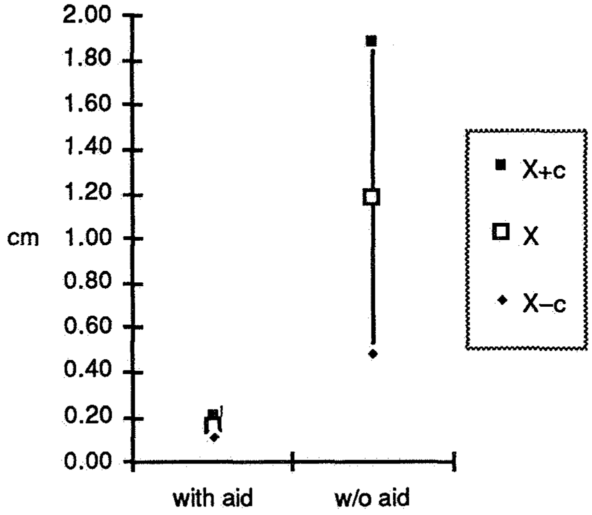


*Fig. 6.5-4 Final Position Error for Subject 2*

Figures 6.5-5 and 6.5-6 show the orientation error measure. As with the position error, both subjects exhibited less error and less variation with the aid. The confidence ranges do not overlap for either subject, again indicating a significant difference related to the presence of the operator aid.

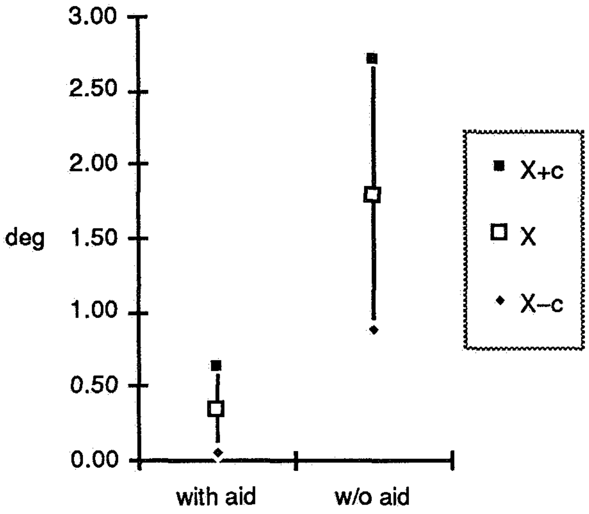


*Figure 6.5-5. Final Orientation Error for Subject 1*

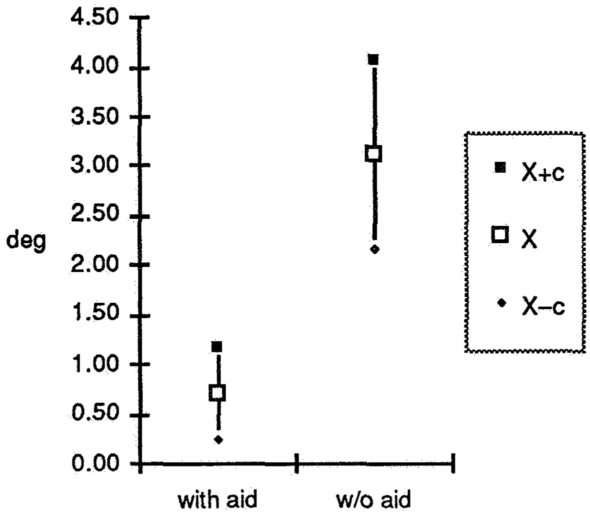


*Figure 6.5-6. Final Orientation Error for Subject 2*

## 6.6 Discussion

The subjects made use of geometric features in the images to position and align the connector, they did not just use the image of the connector halves. The accuracy of the final positions would probably have been worse if the vision target had not been visible in the wrist camera view.

The movement style of subject 1 was smoother than that of subject 2, which apparently had an effect on the effectiveness of the operator aid. The aid display dropped out for approximately a second whenever the vision system lost track of the targets. This track loss occurred whenever one target reached the screen edge, or when the image motion between frames was large. The relatively fast motions of subject 2 caused such track losses in seven of the eight trials. This degraded the time performance of the task due to the momentary freezing of the wrist camera view in the current implementation. This may explain why subject 2 took longer to perform the task with the aid present. Subject 1 on the other hand, only experienced one track loss in one trial. It is possible that if the camera view had remained live, with only the aid freezing during a track loss, the average time for subject 2 would have been lower.

In general the accuracy was much better with the aid present, with less definite results for the task time. Since only two subjects were used, no conclusions about the effectiveness of the aid for a general population can be drawn. Based on the results of this experiment however, further study to support more general conclusions appears merited.

## 7. CONCLUSIONS

In this paper we have described a machine vision based teleoperation aid that improves the operator's sense of perception in performing remote robotics tasks. We have implemented the system and in a preliminary experiment with human operators, have found a significant improvement in their positional accuracy when the aid is used. The aid should be of great benefit to tasks where high accuracy is required, but where there are insufficient camera views, reference markings, etc, to help the operator. The vision based aid also has additional advantages of a large range of operation (in our setup, depths up to 72 cm and yaw angles from [-25°.. 40°]), it is non-contact, and it is integrated with the operator's normal live camera view. Although relatively low cost processing hardware is used, the machine vision system achieves a fairly high update rate of between 5-10 Hz.

This preliminary work has shown the viability of the vision based teleoperator aid and has indicated that there is a great payoff in its use. Future work will follow three directions: (1) improvements in the machine vision system, (2) improvements in the operator's display system, and (3) performance of more extensive experiments.

Improvements to the machine vision system will increase its accuracy, robustness, and flexibility. Specifically, the system currently makes use of a single (wrist) camera view, and locates specially designed optical targets placed on the object. Future enhancements will allow the use of more than one camera view to improve accuracy, and allow a larger set of visual features to be used, which will increase its flexibility and robustness.

Planned improvements to the operator interface include: (1) overlay of the graphical and numerical aids using the IRIS workstation, which will provide live video during track loss and allow positioning of the aids on the entire screen; (2) touchscreen interaction with the vision system to designate target sets of interest in a multi-object view; and (3) stereo presentation of the aids overlaid on stereo video display.

Finally, more comprehensive experimentation will be performed to determine the benefits of different presentations of the vision-derived information, in conjunction with more complex tasks. Communications bandwidth limitations call for a comparison of the relative performance of two orthogonal views, a stereo view, and a single view with aiding.

## ACKNOWLEDGEMENTS

The authors would like to thank the test subjects for participating in the experiments. This research was supported by Martin Marietta IR&D projects D-11R and D-75D.

## REFERENCES

[Bose90] Bose, Chinmoy B., and Israel Amir, "Design of Fiducials for Accurate Registration Using Machine Vision," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 12, No. 12, December 1990, pp. 1196-1200.

[Bejc80] Bejczy, A., J. Brown, and J. Lewis, "Evaluation of "Smart" Sensor Displays for Multidimensional Precision Control of Space Shuttle Remote Manipulator," *Proc. 16th Annual Control Conf., Manual Control*, 1980, pp. 607-627.

[Chan89] Chandra, T., and M. A. Abidi, "A New All-Geometric Pose Estimation Algorithm Using a Single Perspective View," *SPIE Vol. 1192 Intelligent Robots and Computer Vision VIII: Algorithms and Techniques (1989)*, pp. 318-329.

[Crai89] Craig, J., *Introduction to Robotics, Mechanics and Control*, Second Edition, Addison-Wesley Publishing Co., Reading, Massachusetts, 1989.

[Gatr91] Gatrell, Lance B., William A. Hoff, Cheryl Sklair, and Michael Magee, "Robust Image Features: Concentric Contrasting Circles and Their Image Extraction," submitted to *Computer Vision and Pattern Recognition*, June 1991.

[GSFC87] Goddard Space Flight Center, "Robotic Assessment Test Sets," GSFC Report No. SS-GSFC-0029, March 1987.

[Hart88] Hartley, Craig S., and Robert Pulliam, "Use of Heads-up Displays, Speech Recognition, and Speech Synthesis in Controlling a Remotely Piloted Space Vehicle," *IEEE Aerospace and Electronic Systems Magazine*, Volume 3, No. 7, July 1988, pp. 18-26.

[Hir89] Hirzinger, G., Heindl, J., and Landzettel, K., "Predicitve and Knowledge-Based Telerobotic Control Concepts," in *Proceedings of 1989 IEEE International Conference on Robotics and Automation*, pp. 1768-1777, May 1989.

[Hung85] Hung, Y., P. Yeh, and D. Harwood, "Passive Ranging to Known Planar Point Sets," *IEEE International Conference on Robotics and Automation*, March 25-28, 1985, pp. 80-85.

[Kim87a] Kim, W.S. et al., "Quantitative Evaluation of Perspective and Stereoscopic Displays in Three-Axis Manual Tracking Tasks," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-17, No. 1, pp. 61-72, January 1987.

[Kim87b] Kim, W.S., Tendick, F., and Stark, L.W., "Visual Enhancements in Pick-and-Place Tasks: Human Operators Controlling a Simulated Cylinderical Manipulator," *IEEE Journal of Robotics and Automation*, Vol. RA-3, No. 5, pp. 418-425, October 1987.

[Kim89] Kim, W.S., and Stark, L.W., "Cooperative Control of Visual Displays for Telemanipulation," in *Proceedings of 1989 IEEE International Conference on Robotics and Automation*, pp. 1327-1332, May 1989.

[Kris90] Krishnan, Radha, H. J. Sommer III, and Peter D. Spidaliere, "Monocular Pose of a Rigid Body Using Point Landmarks," submitted to *Computer Vision, Graphics, and Image Processing*, 1990.

[Mas89] Massimino, M.J., Sheridan, T.B>, and Roseborough, J.B., "One Handed Tracking in Six Degrees of Freedom," in *Proceedings of 1989 IEEE International Conference on Systems, Man, and Cybernetics*, pp. 498-503, November 1989.

[Pepp83] Pepper, R., R. Cole, E. Spain, and J. Sigurdson, "Research Issues Involved in Applying Stereoscopic Television to Remotely Operated Vehicles," *Proc. of the Int'l SPIE*, Vol. 402, Geneva, Switzerland, April 1983, 170-174.

[Skla90] Sklair, Cheryl, Lance Gatrell, William Hoff, and Michael Magee, "Optical Target Location Using Machine Vision in Space Robotics Tasks," *Proceedings of SPIE Symposium on Advances in Intelligent Systems*, November 1990, in press.

[Tsai87] Tsai, R., "A Versatile Camera Calibration Technique for High-Accuracy 3D Machine Vision Metrology Using Off-the-Shelf TV Cameras and Lenses," *IEEE Journal of Robotics and Automation*, vol. RA-3, no. 4, pp. 323-344, August 1987.

## APPENDIX 1

In our setup, there was no rotation between the station frame and the tool frame in the goal position; i.e., ${}^{T}_{S}T$ was just a translation and ${}^{T}_{S}R = I$. We initially recorded the camera to object transformation when the arm was in the goal position. We then multiplied the current camera-to-object transformation by the inverse of the recorded transformation to yield the pose error. The meaning of this transformation is as follows:

$$\begin{aligned}\left[{}^{C}_{S}T\right]^{-1}\,{}^{C'}_{S}T &= \left[{}^{C}_{T}T\;{}^{T}_{S}T\right]^{-1}\,{}^{C'}_{T'}T\;{}^{T'}_{S}T \\ &= {}^{S}_{T}T\;{}^{T}_{C}T\;{}^{C'}_{T'}T\;{}^{T'}_{S}T\end{aligned}$$

The camera is rigidly mounted on the wrist, so the above equation reduces to:

$$\left[{}^{C}_{S}T\right]^{-1}\,{}^{C'}_{S}T = {}^{S}_{T}T\;{}^{T'}_{S}T$$

We now rewrite the above with the 4x4 homogeneous transformation matrices:

$${}^{S}_{T}T\;{}^{T'}_{S}T \;=\; \left(\begin{array}{c|c} {}^{S}_{T}R & {}^{S}P_{TORG} \\ \hline 0 & 1 \end{array}\right)\left(\begin{array}{c|c} {}^{T'}_{S}R & {}^{T'}P_{SORG} \\ \hline 0 & 1 \end{array}\right)$$

$${}^{S}_{T}T\;{}^{T'}_{S}T \;=\; \left(\begin{array}{c|c} {}^{T'}_{T}R & -{}^{T'}_{T}R\;{}^{S}P_{T'ORG}+{}^{S}P_{TORG} \\ \hline 0 & 1 \end{array}\right)$$

The correct translational component should be $-{}^{S}P_{T'ORG}+{}^{S}P_{TORG}$. Our translational component has the additional factor of the rotation, ${}^{T'}_{T}R$.

# A Color-Coded Vision Scheme for Robotics

Kelley Tina Johnson
NASA/Goddard Space Flight Center
Robotics Data Systems & Integration Section
Code 735.1
Greenbelt, MD 20771

## ABSTRACT

Most vision systems for robotic applications rely entirely on the extraction of information from gray-level images. Humans, however, regularly depend on color to discriminate between objects. Therefore, the inclusion of color in a robot vision system seems a natural extension of the existing gray-level capabilities.

This paper discusses a method for robot object recognition using a color-coding classification scheme. The scheme is based on an algebraic system in which a 2-dimensional color image is represented as a polynomial of two variables. The system is then used to find the color contour of objects. In a controlled environment, such as that of the in-orbit space station, a particular class of objects can thus be quickly recognized by its color.

**Keywords:** Image Processing, Image Algebra, Color, Artificial Intelligence, Vision

## INTRODUCTION

The continued evolution of robotics in America's space program is critical to the advancement of space science. Robots have contributed to the launching of satellites, the collection of lunar surface samples, and a wide array of other NASA activities. As the Space Station Freedom develops in the 1990's, robotics will continue to play a major role.

In response to the anticipated need for routine assembly and maintenance tasks required by the space station, NASA has outlined plans for an advanced robotic system, the FTS (Flight Telerobotic Servicer), to aid in these tasks. Initially, the FTS will assist the astronauts in building the space station. Later, it will perform routine maintenance.

Space Station servicing will require a considerable amount of work to be done outside the spacecraft. The extreme dangers and cost of astronauts performing EVA (extra-vehicular activity) prohibit a repair force from being 100% human. Robotics will necessarily have to be incorporated as a substitute for human labor. Designed correctly, the robot can perform the tasks as capably as a human, but without the same health risks.

The effectiveness of a robot in a particular situation can often be accredited to the control of its human-like appendages. Just as a human relies on arms, legs, feet, hands, etc. to accomplish tasks, robots also have limbs with which to maneuver and to work. However, what is generally overlooked when discussing robotics is the required sensory input a robot needs to remain productive in its environment.

Humans benefit from continuous interaction with sound, light, temperature, etc., but a robot must be specifically adapted to sense and to understand these same stimuli. Because humans rely on vision more than any other sensory input, vision should be a major concern for robotics researchers.

Unfortunately, duplicating human vision capabilities for a robot is not a trivial task. The massive parallel structure of the brain makes current SISD (single instruction, single data stream) digital computer technology obsolete. Even state-of-the-art parallel supercomputers cannot match the processing power of the brain's massive neural structure. Nevertheless, the digital computer is undeniably the success story of the century. Its computational finesse has manipulated data for thousands of applications, ranging from household financial planning to missile guidance. The computer, however, can do what a human cannot, fast numerical computation.

Yet, as machines begin to require more human characteristics, such as vision and natural language processing, it has become necessary to re-evaluate expectations. Number crunching might temporarily have to take a back seat. All of a sudden computers are needed to solve problems for which the current systems are unsuited. The world desires another type of machine that can solve problems like a human.

This enormous opportunity has been seized by Artificial Intelligence researchers. Historically, AI has sought to understand how the human brain functions in order to model it "artificially." It is exactly this expertise which brings AI personnel to the forefront of robotics vision research. How to detect and understand sensory stimuli has become a paramount issue. The success of current and future robotic systems, like the FTS, depends on it.

## Color as an Additional Parameter

Plans for the FTS vision system include its ability to track and recognize objects as well as visually inspecting the success or failure of robot and human operations. AI vision will be the "eyes" of the FTS and, like real eyes, the recognition of objects can be achieved through edge detection. (Marr, 1982)

An edge detector is used to identify the outline, or contour, of an image. The resulting "edged" image reduces the amount of less significant data while extracting the most significant features. The new data set is in a more compact format allowing for swifter image analysis.

Typically, model-matching has been a common technique utilized for identifying an object. This method employs a knowledge base which has previously stored the edged data for many known objects. The data might include features such as length, number, and orientation of edge elements, etc. By producing the edged data for an object and comparing it to the knowledge base, similarities and differences can be found. Given that enough features are determined to "match," the object can be identified with reasonable certainty.

It is desirable for the robot to have an "intelligent" means for reducing the search space. If a single unknown object must be compared against every object in the knowledge base to determine the best possible match, then as the number of objects in the knowledge base increases, the comparison becomes excessively time-consuming and expensive. It is, therefore, desirable to expedite the search process by reducing the search space.

Striving toward this goal, one promising avenue of research involves adding color information to the knowledge base. The requirements for a color system increases the complexity of the hardware when compared to a gray-level system. However, the value of reducing the search space far outweighs this cost.

The proposed system requires color-coding the objects the system will view. In the case of the FTS, this could mean coloring an ORU (Orbital Replacement Unit) red, a thermal utility connector yellow, etc., or specific parts of the objects might be colored or given certain combinations of colors. This new information for each object provides an additional parameter which the model-matcher can use to better discriminate between objects.

Effectively, color becomes the primary feature used for classification. The vision system will now use its apriori knowledge of what it is seeking to converge upon a solution more quickly. If a red object is detected, the system will search only through known red objects for a match. The multitude of other colored objects are ignored and can be immediately eliminated as likely possibilities.

## Image Algebra

AI researchers who seek to design vision systems are necessarily reliant on the quality of low-level information that can be provided. This information is achieved by analyzing unknown input data in terms of known facts contained in the knowledge base. The correctness of the previously determined facts in the knowledge base directly affects the higher-level reasoning of a vision system as does the precision of the input data. Conclusions based on relations between the input data and the knowledge base are key to image understanding.

Image Algebra offers a standardized method for storing and manipulating image information in a knowledge base. The strength of this combination lies in its uniformity: the image is always stored and manipulated in the same way,..*algebraically*. Additionally, the characteristics of Image Algebra provide an excellent means for performing edge detection on an image. In keeping with the ultimate goal of image understanding, edge detection plays an important role.

Most importantly, color images can be processed to find color edges. Compared to gray-level, color edge detection offers a new level of information which can be exploited by a vision system. Overall, this field of study is constantly a source of new techniques for dealing with particularly vexing problems.

## Defining an Image using Image Algebra

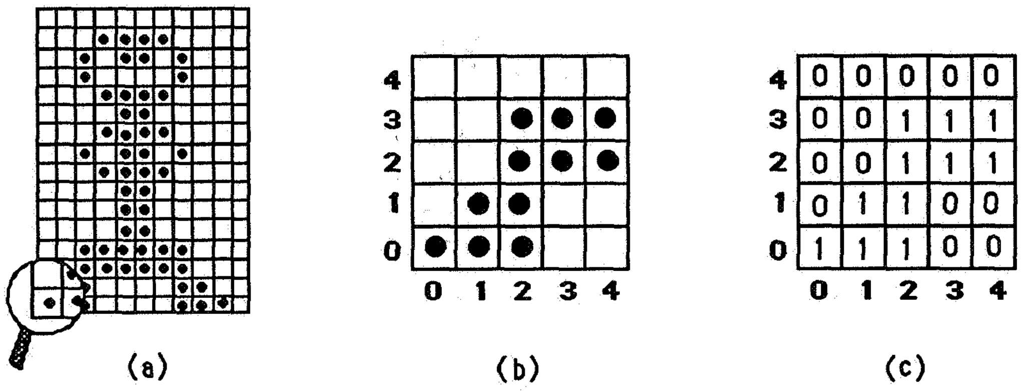
Digital images are typically represented in discrete form as a 2-dimensional array of pixel values. Each pixel corresponds to an (x,y) location in the image. When considering a gray-level image, these values correspond to the intensity, or brightness, of each pixel. For example, if an image spanned 0..255 gray-levels, 0 could represent black, while 255 would be the brightest, being white. Extending this representation to accommodate color will be explained momentarily.

Image Algebra uses a polynomial of two variables, 'x' and 'y', to define a 2-dimensional image. Similarly, a polynomial of three variables, 'x', 'y', and 'z', is used when defining a 3-dimensional image. However, only 2-dimensional images are considered in this paper.

Constructing a polynomial for an image requires only spatial domain information. The complete polynomial will consist of N x M terms, corresponding to the number of rows and columns in the image. The exponents of the 'x' and 'y' variables correspond to the (x,y) location in the image. The coefficient of each term is representative of the pixel value at the location designated by the exponents. This could be either an intensity/brightness or color value depending on whether a gray-level or color image is being defined.

For simplicity, consider a binary image as seen in Fig. 1(a). A binary image limits the pixel values to being either black (1), or white (0). As shown, the grid defines the (x,y) location of the pixels, while the contents of the grid, a black mark or nothing, is symbolic of the value. For this example only, it is assumed that the object is black on a white background.

Fig. 1(b) is a blow-up of the lower left-hand corner of Fig. 1(a). Here the (x,y) coordinates have been added for reference. This corner, (0,0), is considered to be the origin of the image.



|  |  |  |  |  |  |
|---|---|---|---|---|---|
| 4 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 1 | 1 | 1 |
| 2 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 |
|  | 0 | 1 | 2 | 3 | 4 |

(a) (b) (c)

Fig. 1. An example of digital image representation. (a) A digital image. (b) A blow-up of the lower left-hand corner of the image with (x,y) coordinates. (c) Binary pixel values for the lower left-hand corner.

Now, consider the information contained in Fig. 1(c) showing the pixel values. Remember that when constructing the image polynomial, the pixel values are coefficients. Observing Fig. 1(c), it is apparent that many of the terms will be 0. Concentrating on the non-zero terms, the image can be said to be a collection of the following points.

{(0,0), (1,0), (1,1), (2,0), (2,1), (2,2), (2,3), (3,2), (3,3), (4,2), (4,3)}

Using these (x,y) values as exponents, a polynomial expression can be created. It is understood that the coefficient of each term is 1.

$$X^0Y^0 + X^1Y^0 + X^1Y^1 + X^2Y^0 + X^2Y^1 + X^2Y^2 + X^2Y^3 + X^3Y^2 + X^3Y^3 + X^4Y^2 + X^4Y^3$$

Notice how each black pixel is represented according to its (x,y) location. The white pixels have a coefficient of 0 and are not shown because they reduce to zero.

Finally, the polynomial is reduced to a more readable form.

$$1 + X + XY + X^2 + X^2Y + X^2Y^2 + X^2Y^3 + X^3Y^2 + X^3Y^3 + X^4Y^2 + X^4Y^3$$

## Defining a Color Image

When using color, each pixel in the image is assigned a value corresponding to its particular color. Working with eight colors, Fig. 2 below shows the color assignments. Notice that red, green, and blue are assigned to the powers of 2 ($2^0 = 1$, $2^1 = 2$, and $2^2 = 4$). This will become important later.

0 - Black
1 - Blue
2 - Green
3 - Cyan
4 - Red
5 - Magenta
6 - Yellow
7 - White

Fig. 2. Color Assignments

Fig. 3 illustrates a region of a color image containing the colors red, green, and magenta on a black background. Again, for simplicity, only a small region near the origin of the image is displayed. This keeps the exponents small and more readable. In reality, exponents assume very large values due to the considerable size of most images. For example, a 512 x 512 image would require exponents 0..511.

| | | | | | |
|---|---|---|---|---|---|
| 4 | BK | BK | BK | BK | BK |
| 3 | BK | R | G | M | BK |
| 2 | BK | R | G | M | BK |
| 1 | BK | R | G | M | BK |
| 0 | BK | BK | BK | BK | BK |
| | 0 | 1 | 2 | 3 | 4 |

(a)

| | | | | | |
|---|---|---|---|---|---|
| 4 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 4 | 2 | 5 | 0 |
| 2 | 0 | 4 | 2 | 5 | 0 |
| 1 | 0 | 4 | 2 | 5 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| | 0 | 1 | 2 | 3 | 4 |

(b)

Fig. 3. An example color image region. (a) A red, green, and magenta image. (b) Color pixel value assignments using definitions from Fig. 2.

Constructing the polynomial for a color image is very similar to that of the binary case discussed earlier. For example, the image region in Fig. 3 is comprised of the following set of points.

{(1,1),(1,2),(1,3),(2,1),(2,2),(2,3),(3,1),(3,2),(3,3)}

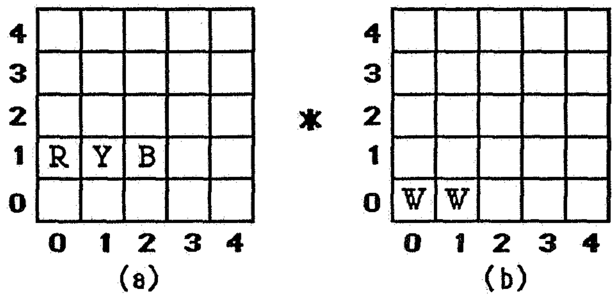
In the binary illustration of Fig. 1, the coefficients were either 0 or 1. Working with color, the pixel values can be 0..7. The color coefficients, representing pixel value, therefore assume the range 0..7. Writing the reduced polynomial with the proper color coefficients yields the polynomial below.

$$4XY + 4XY^2 + 4XY^3 + 2X^2Y + 2X^2Y^2 + 2X^2Y^3 + 5X^3Y + 5X^3Y^2 + 5X^3Y^3$$

## Image Operations

The operations applied to image polynomials may be initially conceptualized as being the addition and multiplication of polynomials according to the standard arithmetic methods.

Consider multiplying the two small images in Fig. 4. The polynomial expressions for the images are also shown.



(a) (b)

Fig. 4. Color Images (a) $4Y + 6XY + X^2Y$ (b) $7 + 7X$

Using standard algebraic technique, the two polynomials multiplied together yield the following result.

$$(4Y + 6XY + X^2Y) * (7 + 7X) = 28Y + 70XY + 49X^2Y + 7X^3Y$$

Clearly, the large coefficients in the resulting equation present a problem. The colors are defined only in the range 0..7, but this equation contains coefficients as large as 70, and no color assignments were created for anything larger than 7.

Redefining the addition and multiplication operations on the color coefficients {0,1,..,7} so that the set is mathematically closed, solves this problem. Closure requires both operations to produce a result which is within the set's limits. In this manner, the coefficients generated will always remain within the range of color assignments.

Fig. 5 shows the new definitions for addition (+) and multiplication (•). The addition table has been produced using bitwise OR while the multiplication table has been produced using bitwise AND (Qian & Bhattacharya, 1991). It should be noted that addition is used to combine similar terms resulting from the multiplication.

| + | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 1 | 1 | 3 | 3 | 5 | 5 | 7 | 7 |
| 2 | 2 | 3 | 2 | 3 | 6 | 7 | 6 | 7 |
| 3 | 3 | 3 | 3 | 3 | 7 | 7 | 7 | 7 |
| 4 | 4 | 5 | 6 | 7 | 4 | 5 | 6 | 7 |
| 5 | 5 | 5 | 7 | 7 | 5 | 5 | 7 | 7 |
| 6 | 6 | 7 | 6 | 7 | 6 | 7 | 6 | 7 |
| 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 |

(a)

| • | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 2 | 0 | 0 | 2 | 2 | 0 | 0 | 2 | 2 |
| 3 | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 |
| 4 | 0 | 0 | 0 | 0 | 4 | 4 | 4 | 4 |
| 5 | 0 | 1 | 0 | 1 | 4 | 5 | 4 | 5 |
| 6 | 0 | 0 | 2 | 2 | 4 | 4 | 6 | 6 |
| 7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

(b)

Fig. 5. Image Algebra Operations (a) Addition (b) Multiplication

Now that the set addition and multiplication are being used, the polynomial operation considered before becomes the following. As expected, the coefficients fall within the defined range.

$$(4Y + 6XY + X^2Y) * (7 + 7X) = 4Y + 6XY + 7X^2Y + X^3Y$$

Especially relevant, is that the operations maintain the integrity of the color combinations. For example, blue and red are known to yield magenta (1 + 4 = 5), green and red yield yellow (2 + 4 = 6), etc. Also, a color added to itself, remains the same (3 + 3 = 3).

## Color Edge Detection with Image Algebra

In the previous example of Fig. 4, it was demonstrated how two polynomials could be multiplied together. In essence, the first image/polynomial is very large. Images are usually at least 512 x 512 pixels. The second image/polynomial is usually only a few pixels. This small polynomial is often distinguished as being an *operator* or *image operator.* When the operator is created specifically for the purpose of edge detection, it is often referred to as an *edge detector operator*, or simply an *edge detector.*

Edge detection is a result of constructing the pixels in the image operator in such a way that an edge/contour is produced when it is multiplied with the original image. The image operator usually includes the term '1' ($X^0Y^0$), plus other terms which will shift the image 1-pixel away from its original position in various directions. For example, X would shift the image 1-pixel in the X direction, while XY would shift the image 1-pixel in the XY direction. This process smears the image. Next, the original is subtracted from the smeared image to remove the interior of the region. This leaves the desired outline of the object. The subtraction is performed using a bitwise XOR.

Consider the edge detector shown in Fig. 6 which contains coefficients corresponding to white(7) (Qian & Bhattacharya, 1991). This operator consists of a 3 x 3 square centered around the origin. The negative exponents assure that every shift in the positive direction, is offset by another in the negative direction. In this manner, all edges regardless of orientation, will be found.

| | | | | | | |
|---|---|---|---|---|---|---|
| 4 | | | | | | |
| 3 | | | | | | |
| 2 | | | | | | |
| 1 | 7 | 7 | 7 | | | |
| 0 | 7 | 7 | 7 | | | |
| -1 | 7 | 7 | 7 | | | |
| | -1 | 0 | 1 | 2 | 3 | 4 |

$$7 + 7X + 7XY + 7Y + 7X^{-1}Y + 7X^{-1} + 7X^{-1}Y^{-1} + 7Y^{-1} + 7XY^{-1}$$

Fig. 6. Image Operator for Color Edge Detection

An example of this operator being applied to an image is shown in Fig. 7.

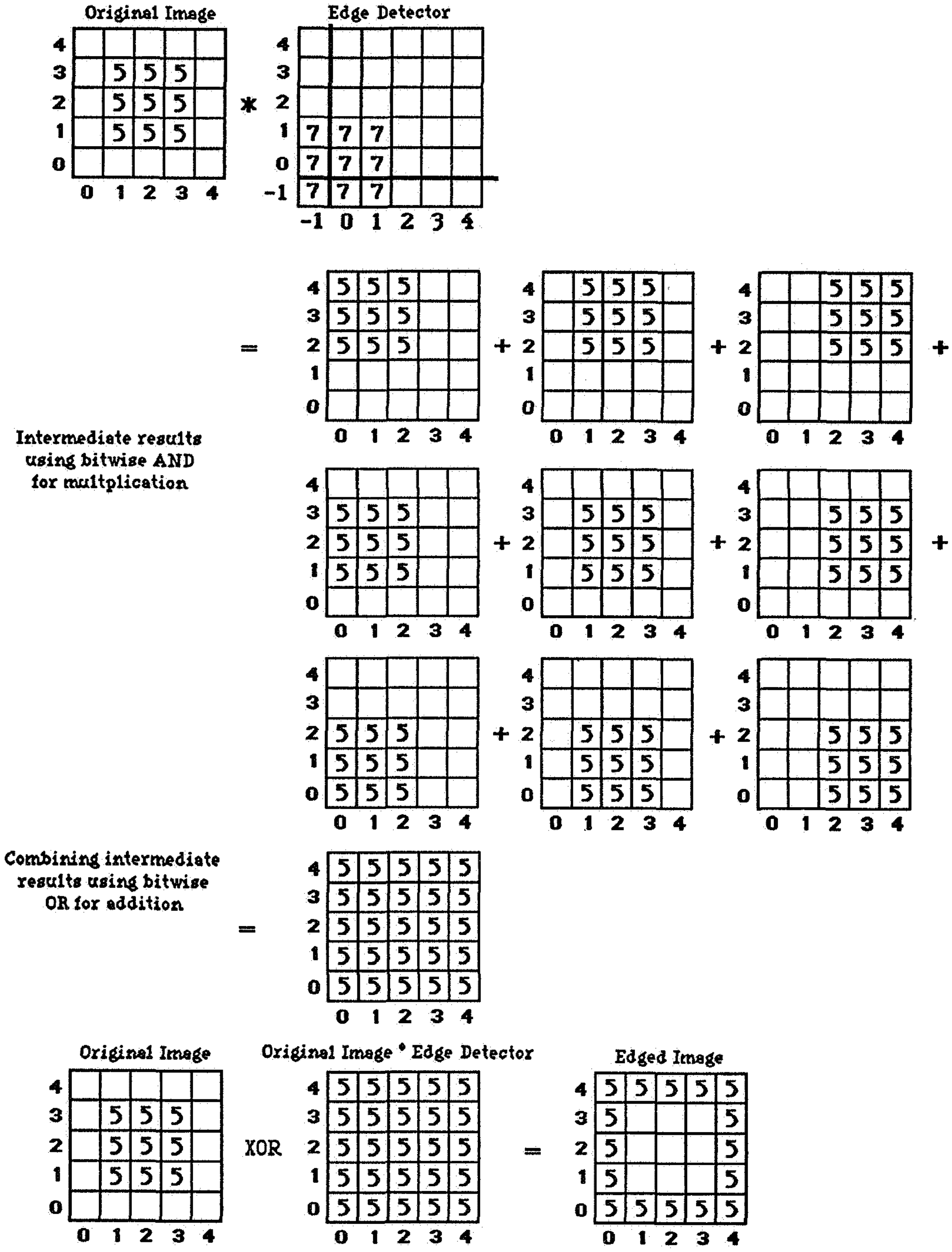

Original Image
Edge Detector
*
=
+
Intermediate results using bitwise AND for multplication
Combining intermediate results using bitwise OR for addition
Original Image
Original Image * Edge Detector
XOR
Edged Image
0 1 2 3 4
-1 0 1 2 3 4

Fig. 7. An example of the color edge detector being applied to a color image.

The strength of this approach is that the coefficients of the edge operator can be changed so that only edges of a certain color are found. The advantage this offers is evident when again considering the requirements for a vision system's knowledge base. Edge detection reduces the data while maintaining the most important image features. When only specific color information is extracted, even more unwanted data is able to be discarded, while more relevant data is brought to the attention of the AI system.

Consider the color assignments in Fig. 8.

| $2^2$ | $2^1$ | $2^0$ |
|---|---|---|
| Red | Green | Blue |

Fig. 8. Primary Color Assignments

All colors that can be displayed on a monitor are created by varying intensity levels of the three electronic primary components, red, green, and blue. Certainly, no system has the capacity to display all potential colors. The number of colors that can be displayed is dependent on the number of bits representing each primary color.

Using 3 bits, 1 for each primary color, 8 unique colors can be generated. Because the 3 primary colors were assigned to the powers of 2, the 8 resulting combinations correspond with those in Fig. 2. This is due to the fact that the particular 8 colors are formed from simple combinations, none of the colors (black), all of the colors (white), one of the colors (red, green, or blue), or two of the colors (magenta, cyan, or yellow).

The binary representation of the eight colors can be seen in Fig. 9. A '1' means that the primary color in that position is at full intensity. A '0' means that that color is not present. For example, cyan is composed of green and blue, but not red. This is shown as a '1' in both the green and blue bits, but not the bit which corresponds to red.

000 - Black
001 - Blue
010 - Green
011 - Cyan
100 - Red
101 - Magenta
110 - Yellow
111 - White

Fig. 9. Binary Color Assignments

Consider again the example presented in Fig. 4. This image was represented by the polynomial below.

$$4Y + 6XY + X^2Y$$

The coefficients represent the color assignments. Presented here in integer form, the same equation could also be written in binary form as shown below. 4 is written as $100_2$ in binary, 6 as $110_2$, etc.

$$100_2Y + 110_2XY + 001_2X^2Y$$

Going back to the edge detector operator, it is seen that the coefficients are all 7, or $111_2$. The result of using 7 for the coefficients is that all edges are found in all three color planes, red, green, and blue. If the coefficient were either red (4), green (2), or blue (1), only the edges of that color would be found. If, on the other hand, a magenta (5 or $101_2$) was used, red and blue edges would be found, but not green. Magenta's binary representation does not have a '1' in the middle/green bit.

Because color cannot be reproduced for this publication, the following examples are provided to demonstrate the edge operator. First, consider the image in Fig. 10. This example shows an image containing seven shapes. The background is assumed to be black (it is not shaded for readability purposes). Assume the image is multiplied by the edge detector operator. The coefficients of the operator are all '7', $111_2$ in binary. Therefore, all edges of all colors will be found because there is a '1' at each primary color bit. All edges will be the color of the object they correspond with, i.e. the red object will produce a red edge, the yellow object a yellow edge, etc.

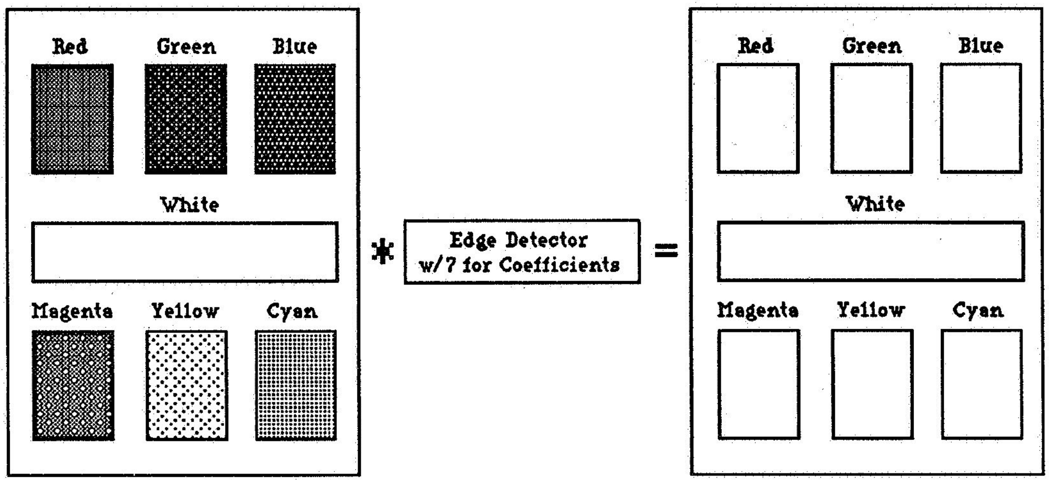


Fig. 10. Example of Edge Detection for All Colors

Now consider the same image, but multiplied by the edge detector with coefficients of 4, red. In this case, only objects which contain red, regardless of other colors, will be detected. As seen in Fig. 11, the red, white, magenta, and yellow objects have been found. These colors all have a '1' in the red color bit. In this case, the edges will all be red, not the color of the object.

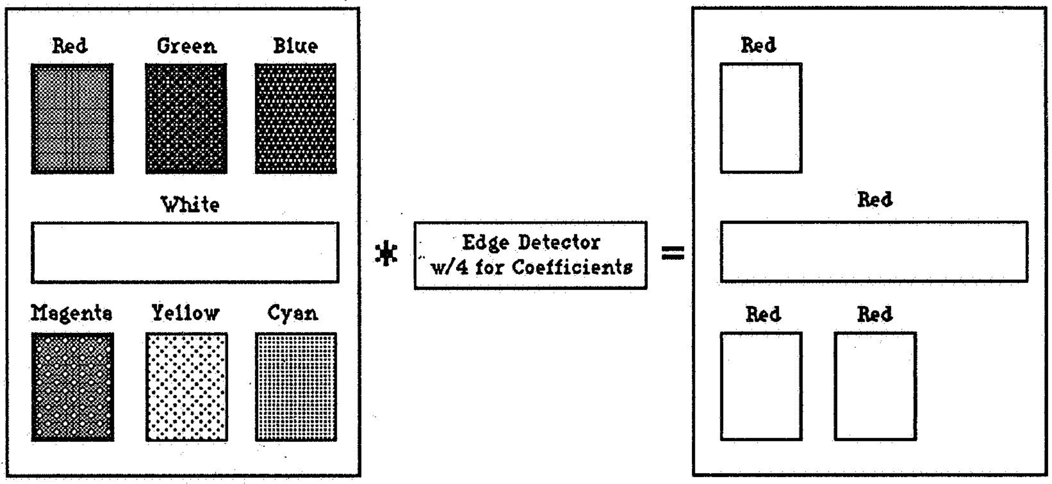


Fig. 11. Example of Red Edge Detection

So far, this publication has only discussed color image operators that have been applied to objects on a black background. Working with colors on a non-black background is not always as straightforward. Consider the case shown in Fig. 12 where a red object is on a blue background. When the edge detector is applied, the result is a blue colored edge inside a red colored edge. At first glance, the edges might seem reversed from what is expected (blue inside red, instead of red inside blue). However, because the edge of an object is comprised of those pixels immediately external to its boundary, the corresponding red and blue edges are indeed in the right place.

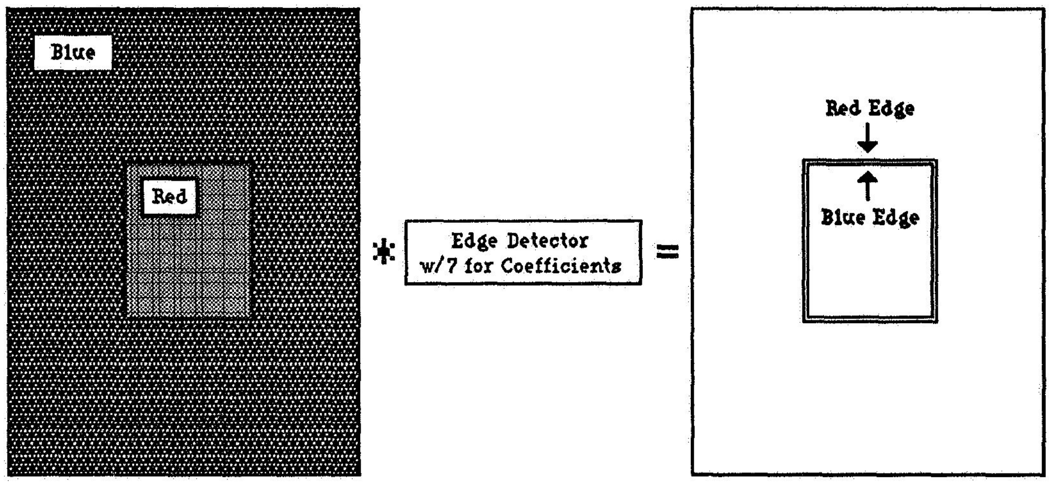


Fig. 12. Edge Detection Performed on a Red Object with a Blue Background

As mentioned earlier, working on a non-black background is a considerably more complicated problem. Two colors adjacent to one another do not always produce edges whose colors can be anticipated. The mixture of red and blue produces the expected red and blue edges only because their bit representations do not have any bits in common, red being $100_2$ and blue $001_2$. If two colors are adjacent to one another *and* have bits in common, the shared bit will be canceled out during the algebraic operations. For example, yellow ($110_2$) and magenta ($101_2$) share the red bit, so the edges produced will be green ($010_2$) and blue ($001_2$) because the red bit is canceled. This feature of color Image Algebra will continue to be a topic for future research.

## Robotics Application

Vision capabilities for robotic systems are critical to establishing autonomous decision-making. For example, NASA engineers expect to supervise the FTS from Earth as well as from space. It is, therefore, critical that the robot be able to respond to as many situations as possible without requiring human assistance. Transmitting communication at every turn would quickly become a burden for both the robot and the human.

The grand scheme for incorporating color Image Algebra into a robotic system like the FTS includes color-coding the objects the robot is expected to interact with. If the objects were to be the same color as the space station truss, or black like the dark space background, little contrast would be available to a vision system. On the other hand, by coloring the objects, not only will they stand-out against the background, but they will be more easily recognized by the knowledge base because of the additional color information.

The technique described throughout this paper obviously lends itself to this robotic application. The Image Algebra approach offers both speed and accuracy. Additionally, the knowledge base of object data can be built up to accommodate classes of objects corresponding with object-oriented design. Similarly, the number of colors can also be increased so that there is a 1-to-1 correspondence between how the image is physically stored and how it is algebraically represented.

## CONCLUSIONS

Image Algebra and color combine as an innovative approach to fulfilling image processing requirements for an AI-based vision system. The algebraic approach provides a solid foundation for image representation as well as a method for edge detection. Color has also proven itself as a useful parameter for discriminating between objects in a scene. The ease and flexibility of this approach makes it a valuable image analysis technique for robotic space applications.

## ACKNOWLEDGEMENTS

The author would like to acknowledge the work of Dr. Prabir Bhattacharya at the University of Nebraska, Lincoln whose work (NAG 5-1382) formed the foundation for this paper. Thanks to Tom Winkert and Colleen Hartman of Code 735.1 NASA/Goddard Space Flight Center for their good humor throughout this project. Also, thanks to Bob Cooke of Cooke Publications, Ltd. for a copy of MathWriter, a scientific word processor for the Macintosh.

## REFERENCES

Marr, D. (1982) *Vision*, Freeman, San Francisco, Calif.

Qian, Kai, & Bhattacharya, P. (1990, June) Binary Image Processing By Polynomial Approach, *Pattern Recognition Letters 11*, 395-403.

Qian, Kai, & Bhattacharya, P. (1991) An Algebraic System for Operating On Gray, Color and 3-D Images. Submitted for Publication.

# EXPERT OPERATOR'S ASSOCIATE:[*] A Knowledge Based System For Spacecraft Control

Mogens Nielsen, Klaus Grue
CRI A/S, Birkerød, Denmark

François Lecouat
MATRA ESPACE, Toulouse, France

January 30, 1991

## ABSTRACT

This paper presents the Expert Operator's Associate (EOA) project which studies the applicability of Expert Systems for day-to-day space operations. A prototype Expert System is developed, which operates on-line with an existing spacecraft control system at the European Space Operations Centre, and functions as an "operator's assistant" in controlling satellites. The prototype is demonstrated using an existing real-time simulation model of the MARECS-B2 telecommunication satellite.

By developing a prototype system, it is examined to which extent the reliability and effectiveness of operations can be enhanced by AI based support. In addition the study examines the questions of acquisition and representation of the "knowledge" for such systems, and the feasibility of "migration" of some (currently) ground-based functions into future space-borne autonomous systems.

## 1 INTRODUCTION

Supervising a spacecraft, interpreting the telemetry received, deciding about the correct on-board operational conditions, reasoning about proper corrections, and executing the appropriate control procedures are complex tasks for modern spacecraft. During the launch phase of the spacecraft, specialists may be at hand, who know about the design of the various sub-systems on-board, but in the subsequent operational phases they will usually not be available for immediate consultancy to the operators responsible for the safe conduct in day-to-day monitoring and controlling the spacecraft.

The complexity of modern spacecraft systems, and the resulting high demands to the personnel operating them, concerns about the potential for human errors, and the risk of inaccessibility of the people with appropriate expert knowledge, calls for improved methodologies and environments providing computerised support for monitoring and controlling spacecrafts.

An essential element in the development of such support is the transfer of parts of the knowledge regarding the procedures for controlling the spacecraft and regarding the design of the spacecraft, from the experts who conceived them, to a system which can be used to assist in the work with the spacecraft in the operational phases.

This has been the motivation for the Operations Center of the European Space Agency (ESOC) to define a 3 year study project, called the Expert Operator's Associate (EOA) project, with the aim of developing a prototype expert system for assisting in the operation of satellites. The MARECS-B2 communication satellite has been chosen as an example case for demonstration of the prototype. The prototype will be demonstrated with the aid of ESOC's "high fidelity" real-time MARECS-B2 spacecraft simulator (a software model) which operates in closed loop communication with the ground control system via simulated telemetry and telecommand links.

The project is structured such that the first phase concentrates on constructing the basic system. It is here demonstrated how the operator can be assisted in the selection and execution of Flight Control Procedures (FCP), covering the situations where the spacecraft operates within the limits prescribed by the specifications of the satellite.

[*] The study is performed under Contract No. 7627/88/NL/DG to ESA for ESOC and ESTEC. It is carried out by a consortium composed by CRI and MATRA.

In the second phase of the project, the scope is extended such that some of the situations where the spacecraft does not operate inside these limits, are taken into account. It is demonstrated how to assist in the situations where the problem can be diagnosed immediately, and handled by predefined Contingency Recovery Procedures (CRP).

Finally, several experimental extensions of the system are investigated. One extension is operator assistance in the situations which cannot be immediately diagnosed. Another extension is machine learning, where new knowledge (e.g. CRP's) is developed as a result of a dialogue with a spacecraft expert, and stored in the knowledge base of the system . Furthermore, the question of to which extent control functions can be "migrated" from the ground to future spacecraft, and the question of how to "streamline" the transfer of knowledge from the spacecraft experts to the system, will be addressed.

The prototype developed is a workstation based system, controlling the process of daily operations of the spacecraft. It works in a real-time environment communicating with the spacecraft operator and the spacecraft[1]. The user interaction is facilitated by a graphical user interface utilizing state of the art techniques such as mouse, multiple windows, and pop-up menus.

At the time of writing, the project is near its completion and the paper presents the overall results, concentrating on the knowledge representation used, and the system architecture. In Section 2 the general problem domain and the example case is further described. Then, in Section 3 the functionality and architecture of the prototype system is introduced. In Section 4, the representation and structuring of the knowledge used by the system, and the expert functions is described. In Section 5 it is illustrated how the execution of Flight Procedures is implemented. Alarm processing is described in Section 6. Finally Section 7 concludes and the continuation of the project is detailed further.

# 2 THE PROBLEM DOMAIN AND THE EXAMPLE CASE

The operating state of orbiting spacecraft is monitored and controlled on the ground at ESA's European Space Operations Centre (ESOC) at Darmstadt, W. Germany, by specialist personnel supported by on-line spacecraft control computer systems (at ESOC). These computers receive telemetry data from each spacecraft, typically in near-real-time via ground stations in various parts of the world. Data from the telemetry are evaluated and displayed to the spacecraft controller, who in turn can initiate the uplink of telecommands (via the ground station) to the remote satellite, from his computer console.

MARECS-B2 is a geosynchronous maritime communications satellite, and is an interesting example case for an on-line Expert System to support spacecraft control. MARECS-B2 poses requirements in day-to-day operation, which are typical for the current generation of telecommunications satellites. The downlinked housekeeping telemetry data, flowing 24 hours per day, provides a "snapshot" of the spacecraft state in a "format" of several hundred "parameters" (readings of on-board sensors) every 19.2 seconds.

The spacecraft control computer system for MARECS-B2 is the ESOC Multi-Satellite Support System (MSSS).

The spacecraft controller monitors only a few of the telemetry parameters at any time, but the MSSS performs automatic checks on many of the parameters in each new format when it has been received. If the checks discover that parameters are outside their normal operating range, or status, audible and visual alarms are raised, so that the spacecraft controller is aware of a possible problem and can decide what to do.

In regular day-to-day operation, which is the type of activity which can be most effectively supported by EOA, the actions of the spacecraft controller are, in principle, completely defined by a large manual of operations procedures known as the MARECS-B2 Flight Operations Plan (FOP). This comprises Flight Control Procedures (FCP) covering nominal operations, and Contingency Recovery Procedures (CRP) which describe the actions to be taken in the event of non-nominal cases. The existence of the FOP ensures that operations can be carried out with a high degree of efficiency (speed in effecting configuration changes which affect the end-user services provided by the satellite), and reliability. Both of these aspects are of prime importance in the provision of telecomms services.

Nominal operations of the spacecraft are preplanned, and a schedule is defined a few days beforehand by a specialist, who selects the FCP's required, and defines the time at which they are to be performed. However, it occasionally happens

[1] via the Multiple satellite Support System (MSSS)

that during operation of the pre-planned schedule, the spacecraft exhibits some unexpected behaviour. The spacecraft controller then has the task of selecting the appropriate CRP's. For this, he may need the assistance of a specialist engineer, but the latter may be unavailable immediately (eg. in the middle of the night).

It also affects the complexity of the task that the spacecraft controller must sometimes take account of actions and knowledge about the spacecraft state in the past, ie. historical information.

One of the functions of the EOA is to assist the spacecraft controller in choosing the right CRP's in a given non-nominal situation. In many cases this will be possible on the basis of straightforward matching of the situation to correspond descriptions stored with each CRP. Thus the EOA will effectively speed up the selection process. However, the situation will sometimes arise where the choice of CRP's is uncertain. The study aims to show how the EOA can assist in the selection of recovery action, also in such cases.

# 3 OVERVIEW OF THE EOA

## 3.1 Functions

The core functionality of the EOA system is to assist the spacecraft operator in nominal spacecraft operation. Support is given by:

- receiving, interpreting and displaying information regarding the state of the satellite
- proposing selected procedures based on the current operating state of the spacecraft and the users indication of the desired state
- presenting the chosen procedure to the user in both textual and graphical form
- preparing the various spacecraft command sequences needed for the execution of the plan, and on acceptance from the user, sending them to the MSSS.
- receiving and evaluating reports from the MSSS on commanding activity
- continuously verifying the validity of constraints posed by the procedure

Non-nominal situations are identified by the receival of alarms (e.g. TC verification failure alarms or Out Of Limit alarms) or by trend analysis of telemetry parameters. At present the EOA system assists in processing alarms by:

- investigating the cause of the alarm
- distinguishing alarms requiring action from non-relevant or expected alarms
- invoking and executing Contingency Recovery Procedures (CRP's)

Situations requiring complex diagnosis before execution of a CRP or requiring actions which are not implemented in the form of an existing CRP, are considered in the last phase of the project, and is therefore not yet demonstrated by the prototype.

FCP's and CRP's are both special cases of Flight Procedures. EOA supports the execution of procedures in general, and therefore the execution of CRP's are supported in the same way as execution of FCP's.

The EOA utilizes the knowledge of experts to perform procedures, and to reason about problems in much the same way as is done by the experts themselves. It has the ability to explain its reasoning, and incrementally acquire new knowledge.

Additionally, the EOA is more flexible than conventional software, e.g. it responds opportunistically to incoming data, or situations, and modifies its behaviour under varying conditions. As an example, procedures have to cope with the exigences of the current situation, or cope with reconfiguration or modification of the units considered.

The EOA provides a number of expert functions integrated within the system which can be organized along three axis:

- procedure generation,
- spacecraft state monitoring,
- execution scheduling.

These types of functions are described in more detail in Section 5, and can be summarised as follows:

Concerning *procedure generation*, the functions range from interpretation and execution of already existing procedures to generation of new procedures.

With *spacecraft state monitoring*, functions range from conventional verification of patterns of parameters to complex failure diagnosis.

*Execution scheduling* functions are initially dedicated to controlling the timing of procedure execution. However, it happens that procedures compete for execution and the system provides functions to arbitrate conflicts.

Additionally EOA has facilities for supporting the spacecraft engineer in editing and maintaining the different types of knowledge in the knowledge bases. In particular, a syntax driven Flight Procedure editor has been developed, in addition to standard knowledge maintenance facilities.

## 3.2 System Architecture

The EOA system communicates with two external entities: the user and the MSSS.

The EOA system runs on an independent SUN workstation and communicates with the MSSS system via a X.25 communication link. It is implemented in the programming language Common LISP using the expert system shell KEE. However, in order to ensure proper speed in performance, and to have proper access to the operating system, parts of the system taking care of communication with the MSSS and the user is implemented directly in the C programming language.

The architecture has been designed with special attention to the fact that the EOA is integrated in a real-time environment, and that it must always be able to respond to the MSSS (eg. to process alarms). Furthermore an aim of the architectural design has been to construct an open ended and modular architecture, thereby supporting maintenance and future extensions.

The result is a multiprocess architecture, consisting of a number of interacting systems, communicating through a common protocol. The architecture is outlined in Figure 1.

The EOA MANAGER takes care of the overall scheduling in the system, and the internal communication protocol.

The Dialogue System is a monitor for the User Interface. This interface is described in the next section.

Receiving, buffering and parsing of information from the MSSS, and formatting and sending messages to the MSSS is handled by the External Systems Interface. The TM/TC System is a monitor for the External Systems Interface.

All the system Knowledge Bases have a common interface, called the KB Methods, through which all accesses are made . The Knowledge Base Management System constitutes a monitor for the Knowledge Bases of the EOA, and contains functions for retrieving, inferring and updating knowledge.

The Flight Procedure Execution System is the central system supervising and controlling the execution of procedures (FCP's and CRP's), interpreting TM's, validating and verifying TC's etc.

The Knowledge Maintenance System monitors the execution of the EOA, the user and MSSS interactions, and controls the consistency, completeness and feasibility of the EOA knowledge. The system propose updates of the knowledge and also evaluate updates proposed by the user.

The Fast Response Recovery System takes over control in case of a non-nominal situation and performs alarm processing, and selection of CRP's to invoke in cases where this can be done without complex diagnosis.

In other non-nominal situations control may be transferred to an Advanced Reasoning System which, in close interaction with the user, performs complex diagnosis, and generates new procedures if necessary. As indicated in Figure 1 this system is not implemented in the current prototype so far.

## 3.3 User Interface

There are two main categories of users: *spacecraft operators* who control the daily operations of the spacecraft, and *spacecraft engineers* who are the experts knowing about the design of the spacecraft. Each has a different pattern of communication with the system.

The User interface utilizes "state of the art" Man Machine Interface (MMI) techniques, including mouse, windowing, and pop-up menus in the user interaction. It has been designed using the powerful facilities of KEE.

Figure 2 shows the basic layout of the EOA screen used for daily operations. It is divided into two separate areas, the System Area and the Procedures Area.

In the System Area all system level information and EOA operational information is displayed, and most user dialogue takes place here. In the basic layout, windows for querying the user (Prompt Window), for logging the operations and tests are also as default placed in the system area. However, these can be moved around by the user. The system area contains an array of buttons which are used to select system functions.

In the Procedures Area the active procedures are displayed, and progress of the procedures is monitored. The area contains, for each active procedure a group of windows for the execution

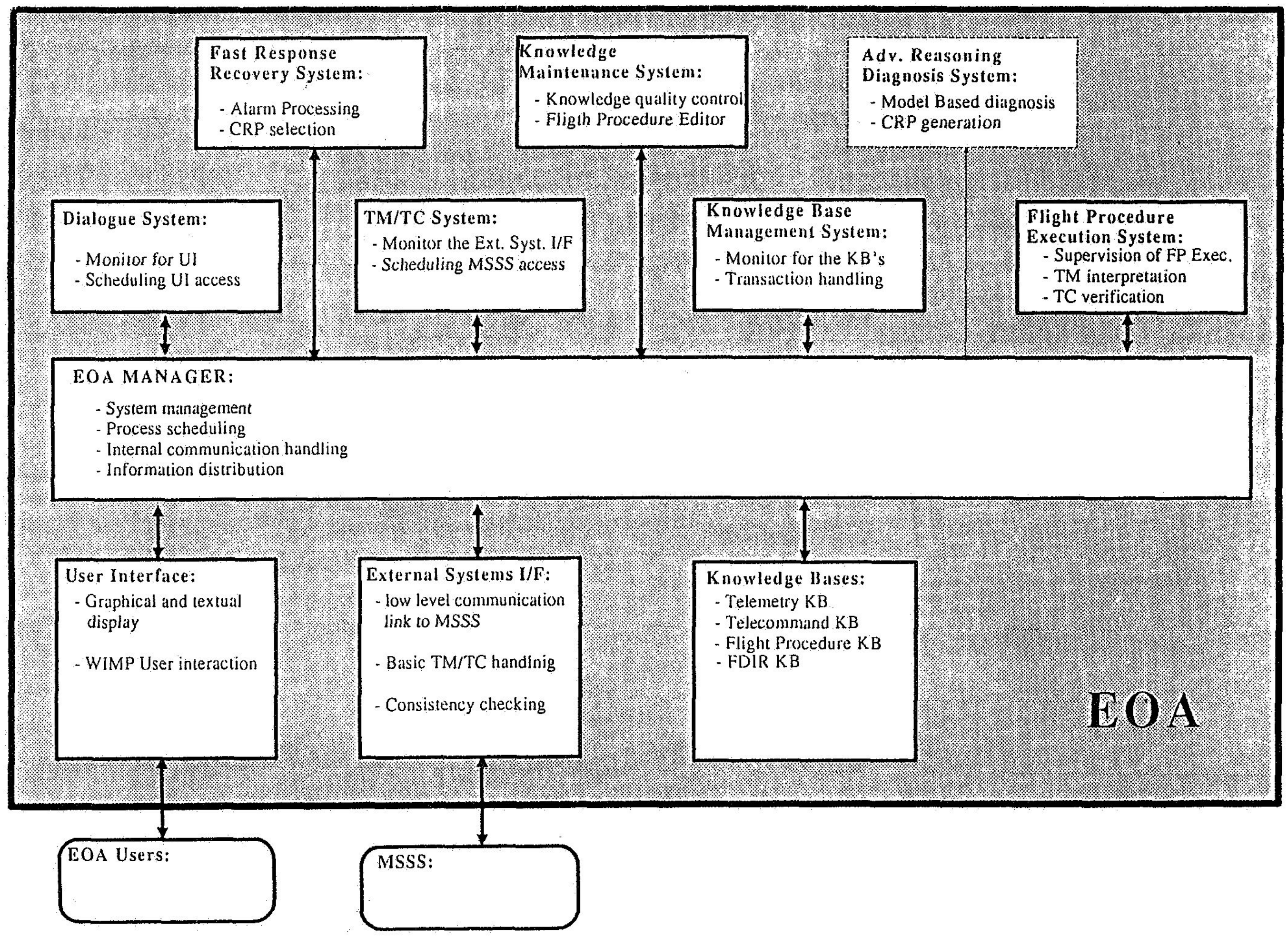
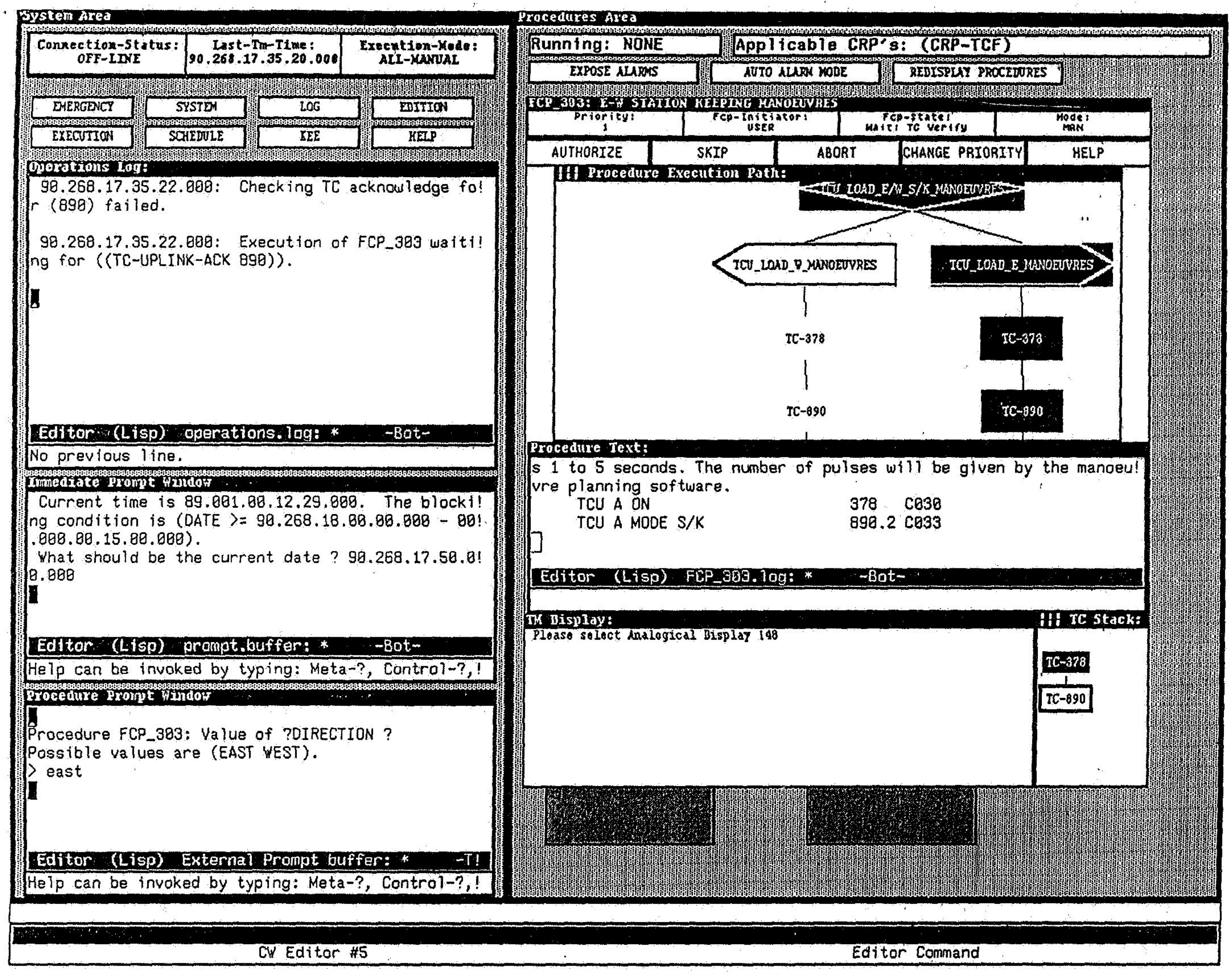


Figure 1: Architecture of the EOA System



```
 90.268.17.35.22.000:  Checking TC acknowledge fo!
r (890) failed.

 90.268.17.35.22.000:  Execution of FCP_303 waiti!
ng for ((TC-UPLINK-ACK 890)).
```

```
Editor (Lisp) operations.log: *      -Bot-
No previous line.
```

```
 Current time is 89.001.00.12.29.000.  The blocki!
ng condition is (DATE >= 90.268.18.00.00.000 - 00!
.000.00.15.00.000).
 What should be the current date ? 90.268.17.50.0!
0.000
```

```
Editor (Lisp) prompt.buffer: *      -Bot-
Help can be invoked by typing: Meta-?, Control-?,!
```

```
Procedure FCP_303: Value of ?DIRECTION ?
Possible values are (EAST WEST).
> east
```

```
Editor (Lisp) External Prompt buffer: *      -T!
Help can be invoked by typing: Meta-?, Control-?,!
```

```
s 1 to 5 seconds. The number of pulses will be given by the manoeu!
vre planning software.
    TCU A ON                      378   C030
    TCU A MODE S/K                890.2 C033
```

```
Editor (Lisp) FCP_303.log: *      -Bot-
```

```
Please select Analogical Display 148
```

```
CW Editor #5                                   Editor Command
```

Figure 2: EOA Prototype System Screen Layout for Daily Operation

and control of that procedure. The procedure is displayed both textually in the Text window, and graphically as a flow diagram in the Procedure Execution Path window, where the part of the procedure already executed appears in highlighted form. There are also windows displaying the information to and from the satellite (TC Stack and TM Display) window. For each procedure, there is an array of buttons for the control of the procedure (authorize an action, skip a step, abort procedure, ...).

Spacecraft engineers can use another EOA screen[2] to display and edit knowledge bases. It includes a syntax driven editor permitting to write procedures.

# 4 KNOWLEDGE REPRESENTATION

The knowledge structure in the EOA has been organized so as to provide satisfactory solutions to the specific problems of procedure generation and execution.

Procedures can be structured as sequences of *steps*. Each step implements a piece of procedure, and contains tests, actions (TC uplink, display message, ...), conditional statements, go-to statements, iterations and transfers to other pieces of informations (e.g. other procedures).

With conventional sequential programming techniques, the order of task execution within a program is entirely determined by the control structure of the code. Conventional programs are rather non-responsive to unanticipated situations, and lack flexibility. The EOA approach is to describe each procedure or part of procedure as a set of schematic instructions (called *scripts*) which are expanded (or interpreted) in the context of the execution. Each schematic instruction describes a *goal* that the system will try to achieve in executing the procedure. An inference mechanism provides a means for directly using the knowledge in the system to reach the desired operational goals, through choices of applicable knowledge.

The declarative semantics which is used together with the inference mechanism provides a good flexibility, allows for incremental changes to the system, and explanatory capabilities.

Another issue which comes with conventional programming techniques is that pieces of code (e.g. subroutines) are named or labelled with arbitrary names which have to be unique. The drawback of this approach is that the link between a piece of code and its functionality may be lost, hereby loosing software engineering and explanatory possibilities. In the EOA, each set of schematic instructions (or scripts) is attached to a name which specifies its goal. This goal is used by the inference mechanism so as to achieve the desired operational goals. Therefore, the EOA is goal-oriented.

However, attaching a goal to a script is not sufficient. There may indeed be many ways to achieve a goal, each way being applicable in a specific context. A *context* is a set of facts which represent the state of the world, as it is affected by the procedure. With conventional programming techniques, it is only the control structure that allows to call a subroutine or another. With the EOA, each script has attached a goal but also a context specifier, which specifies in which circumstances the considered script can be used to achieve the attached goal. During procedure execution, it is the inference engine which identifies for an invoked goal, which script is applicable with respect to the current execution context.

A *context specifier* is defined as a list of variables with desired values. This implementation paradigm has been chosen so as to achieve a double purpose. As explained above, the aim is to provide a deterministic and unambiguous definition of the context where the script is applicable. The list of pairs (variables desired_values) represent the facts which have to be true in the context of the execution. The context specifier may also be needed to query supplementary informations, as needed for the execution of the script. It is possible in the script to specify the context for a call to a goal, or to modify the current context. Each context instruction is lexically scoped. Therefore, the EOA is also context-oriented.

The execution of a piece of procedure described by a script calls for many other informations which are explicitly or also at times implicitly stated in a conventional procedure. These informations have been formalized using the Theory of Plans ([Wil83]). The selected types of informations are:

- *pre-execution checks*, which have to be true before continuing the execution of the piece of procedure,
- *execution constraints*, which have to remain true throughout the execution of the piece of procedure,

[2] by "EOA screen" we mean a specific layout of the workstation display

- *the script itself*, which describes the checks or actions to be performed, and goals to be achieved with this piece of procedure,

- *post-execution checks*, which have to be true immediately after the execution of the piece of procedure,

- *post-execution constraints*, which have to remain true after the execution of the piece of procedure, until they are unset by another piece of information.

It appears then that many informations are attached to a given goal. In the EOA, the chosen implementation paradigm is to group all these informations into an object, whose facets will hold the various types of informations (Figure 3).

As previously explained, several procedures may exist, allowing the achievement of a given goal in different contexts. These procedures may share common pieces of informations. Therefore, they are grouped in a hierarchy, where the parent procedure holds the the common pieces of information, including in particular the goal specification. The common informations are inherited down the hierarchy, until overwritten by local information specific of a procedure or sub-hierarchy of procedures. Procedures are thus implemented as a hierarchical library of objects.

The objective of the knowledge acquisition process is then to feed in each object complete expert knowledge, so that each object contains necessary and sufficient information for object selection and object execution. The sources of knowledge can be spacecraft design (e.g. for the various sub-systems of the spacecraft), or operational experience (e.g. general technical expertise of mission controllers).

To conclude with, the EOA is object oriented, in order to provide an efficient and convenient implementation for procedural expert knowledge.

The EOA project share common goals with PRS ([Geo85],[Geo86]); namely it aims at building a system that explicitly represents and reasons about procedural knowledge. The EOA approach is less general in the sense that control knowledge is not represented as explicitly as in PRS (e.g. the fact that TC failure alarms have priority over COL alarms must be modified by the implementor). On the other hand the EOA language is richer. It permits to implement complex procedures with iterations and conditionals that look quite similar to real procedures. This facilitates manual validation of procedures.

# 5 EXECUTION OF FLIGHT PROCEDURES

This section illustrates how the knowledge implemented in the EOA is used for the execution of flight procedures.

## 5.1 Procedure generation

The system is used in a goal-oriented way. The user can enter a goal corresponding to a procedure or a step. The user is prompted if the specified goal does not match unambiguously one of the goals known by the system.

When one goal is unambiguously identified, its applicability with respect to the current context is examined by the system. To do so, the system collects the object or hierarchy of objects which are able to perform the specified goal. Then, using the context specifier of each object, it tries to find at least one applicable object. The introduction of required information is done through interaction with the user, when information is not available in the system. If no object is found applicable, the system leaves the procedure execution mode and enters a procedure generation mode (to be developed) where the system tries to construct a small procedure in order to set the world in a configuration compatible with the specified goal. This is typically a planning process, with chaining of "operators" from one state to another.

When a convenient object has been found to achieve the initial goal, its execution is initiated. The instructions described in Chapter 4 as pre-execution checks are initially performed. Execution constraints are asserted and checked periodically. The most important part of the execution is with the interpretation of the script. As mentioned in Chapter 4, the script contains a set of instructions, such as actions (Telecommand uplink), checks (e.g. Telemetry Values), conditional statements, and call to other goals. Finally, post-execution check instructions are performed and post-execution constraints are asserted.

In the process of cascading the initial goal into sub-goals specified in the script, the inference engine will recursively try to achieve the sub-goals. Each sub-goal can be called within a specific context, by locally amending some aspects of the current context. The initial goal will be considered as achieved as soon as all goals in the cascade of sub-goals are achieved. Achieving each sub-goal is done sequentially, respecting the procedural order described in the script.

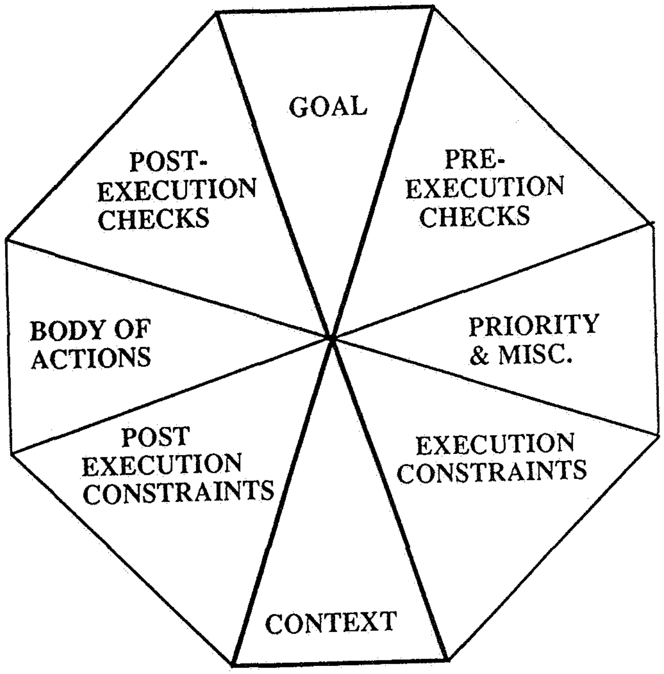


Figure 3: EOA prototype system: Object facets

An interesting feature is that the inference engine which is used to perform script instructions and cascade into sub-goals is based on an interpreter which is interrupt driven. Whenever the execution of instructions finds some reason to stop (e.g. wait for the right time or detection of anomaly), the code which is returned contains:

- the *context* of the current execution,
- the *continuation* of the execution, i.e. all the instructions which are to be executed as soon as the execution resumes.

This is very convenient so as to explain what has been done and what remains to be done.

When something goes wrong in the execution process, the system analyses the cause of the procedure interruption in order to assess which measures are needed to enable the resumption of the execution.

As an example, when the system is waiting for the right time to initiate some action, no special action has to be performed. In the same way, when some telecommand has just been uplinked, and the corresponding telemetry check fails, nothing has to be done since at least one telemetry flow has to come down before the telemetry actually appears as changed. On the other hand, if the telemetry check still fails after reception of several telemetry flows, the system reacts to this anomaly. This may result in uplinking again the Telecommand, or entering a complex diagnostic process. In the same way, when no object is found applicable for the achievement of a given goal, the system may ask the user to confirm the goal specification together with the call context, and later on, initiate the construction of an appropriate new procedure.

## 5.2 Spacecraft State Monitoring

In a first phase, the control of the spacecraft state is done through the verification of the values of a large set of telemetries, generally grouped into Analogical Displays in the MSSS workstations. The EOA does not copy all the TM verification carried out by the MSSS, but focus on verifying selected parameters in order to assess the progress of procedures. The basic reaction to some unexpected telemetry value is described in Section 6.

## 5.3 Execution Scheduling

The execution of procedures may be time driven. For example, the performance of eclipse operations has to comply with a precise timing. The EOA provides functions for time monitoring.

Another type of scheduling problem exists when time constrained procedures compete for execution. Another example is a contingency recovery procedure (CRP) being initiated while a Flight Control Procedure (FCP) is being performed. The EOA could include functions to manage earliest start times (ESD) and latest completion dates (LCD), and implements heuristics to prioritize a procedure against another.

When performing procedure generation, the EOA may also need to manage scheduling aspects in order to verify the feasibility of the generated procedures.

# 6 ALARM PROCESSING

A key skill for spacecraft control is the ability to react quickly to any kind of alarm from the MSSS. Reactions range from simple actions like calling an expert for help, to important decisions like ignoring an alarm or choosing and executing a contingency recovery procedure. For each possible alarm the spacecraft controller can find in the FOP a sequence of actions that can be done. Since these actions have been written conservatively they often lead to a call for help.

One goal of the EOA is to provide assistance to increase reliability, speed and scope of day alarm processing. The implemented prototype deals with alarm combinations that have been foreseen in advance and can be treated by existing emergency procedures.

In compliance with the philosophy of the project it was not attempted to design new alarm mechanisms but to provide assistance to users of the existing control center. The EOA must deal with all the alarms of the MSSS. Nevertheless it is also possible to implement new kinds of alarms. This can be done by one or several procedures running permanently to perform a trend analysis for instance.

The MSSS generates two kinds of alarms:

1. *TC verification failures* when something goes wrong in a TC uplink, and
2. *Out Of Limit (OOL) alarms* when some parameters goes outside some limits defined dynamically on the MSSS or have inconsistent values.

Thanks to the multiprocess capability of the EOA, alarms are processed as soon as they arrive from the MSSS without interrupting procedures. One important problem is to know which alarm to focus on since an alarm rarely occurs alone. For this the EOA uses different kinds of knowledge: priority number on OOLs, alarm prediction included in procedures, mode definitions which can explain that an alarm has been caused by another one. In any case the user can focus on the alarm of his choice and can discard non relevant alarms.

Once an alarm has been selected a set of rules are evaluated to generate a list of all the procedures that can be applied. Each procedure can have such a rule, written in the EOA language, to indicate whether it is appropriate to enter the procedure. If the rules have been well written at most one procedure will be applicable. In the other case the user has to arbitrate which procedure to start or call for help.

The alarm processing scheme described here can be improved in many ways. One of them being the interface with a diagnostics expert system such as DIAMS ([Haz88]) that incorporate spacecraft design knowledge for those situations that require a sophisticated diagnostics method that cannot be implemented conveniently as a procedure.

Another direction for improvement consists in developing a more sophisticated priority scheme. When alarms occur during the execution of the FCP the EOA may have to start a CRP in parallel. The default rule is to give priorities to CRP's over FCP's. Clearly this can be improved since a FCP may have crucial hanging constraints (e.g. put a component off). One solution would be to acquire priority numbers attached to each steps of procedures from experts. A more ambitious scheme would be to infer these priorities from design and operational knowledge.

# 7 CONCLUSION

A large part of the functions described in this paper have already been implemented leading to a prototype that covers assistance to spacecraft controllers for normal daily operations and simple non-nominal situations. Obviously there is room for a lot of improvements and extensions but the EOA has already shown interesting results.

Eight procedures that are quite representative from the MARECS-B2 FOP have been implemented including station keeping, eclipse operations, recovery from payload switch-off, and recovery from automatic reconfiguration. The syntax driven procedure editor and the knowledge base inspectors together with the methodology for procedure generation should permit to extend this set.

A flexible multiprocess architecture for real time expert system makes possible the communication and integration with the ESOC MSSS.

Evaluation of the EOA is carried out in close cooperation with potential users, namely the operators and spacecraft engineers working at ESOC. There are good hopes that the EOA will demonstrate the feasibility and utility of knowledge based operator assistance for spacecraft control.

## References

[Geo85] Michael P. Georgeff. Reasoning about procedural knowledge. In *Proceedings of the AAAI 85 Conference*, pages 41 – 49, 1985.

[Geo86] Michael P. Georgeff. Procedural knowledge. *Proceedings of the IEEE 86 Conference*, Volume 74(Number 10), October 1986.

[Haz88] M. Haziza. An expert system shell for satellite fault isolation based on structure and behaviour. In *Proceedings of the ESA Workshop "Artificial Intelligence applications for space projects"*. ESA ESTEC, November 1988.

[Whe88] J. Wheadon. Expert system for european space infrastructure. In *International Symposium on Europe in Space – The Manned Space System –*, April 1988.

[Wil83] R. Wilensky. *Planning and understanding. A computational approach to human reasoning.* Addison Wesly Publishing Company, 1983.

# *Tools & Techniques*

# VALIDATION AND VERIFICATION OF EXPERT SYSTEMS

Lewey Gilstrap
Computer Sciences Corporation
Beltsville, MD

## Abstract

Validation and verification (V&V) are procedures used to evaluate system structure or behavior with respect to a set of requirements. Although expert systems are often developed as a series of prototypes without requirements, it is not possible to perform V&V on any system for which requirements have not been prepared. In addition, there are special problems associated with the evaluation of expert systems that do not arise in the evaluation of conventional systems, such as verification of the completeness and accuracy of the knowledge base. The criticality of most National Aeronautics and Space Administration (NASA) missions makes it important to be able to certify the performance of the expert systems used to support these missions. This paper presents recommendations for the most appropriate methods for integrating V&V into the Expert System Development Methodology (ESDM) and suggestions for the most suitable approaches for each stage of ESDM development.

## Introduction

Expert systems are mechanizations of the cognitive problem-solving capabilities of human experts. At the outset of expert system development, it is not known whether it is even possible to model and mechanize the mental processes of human experts, much less how long it would take and how much it would cost to do so. The procedures used to develop expert systems are directed toward extracting from the expert the knowledge and skills used in an expert task and toward preparing a series of successively more refined prototypes that mimic the behavior of the human expert. Expert systems tend to evolve rather than follow a planned course of development, and their life cycle differs significantly from the life cycle of conventional systems.

Because of the differences in the life cycles of expert and conventional systems and the need for sound management of expert system development projects, the Data Systems Technology Division of the National Aeronautics and Space Administration/Goddard Space Flight Center (NASA/GSFC) 4 years ago undertook a program to formulate a methodology for expert systems to be developed at GSFC. The result of this effort was the Expert System Development Methodology (ESDM) (CSCa, 1988; CSCb, 1988; CSCc, 1988; Sary et al., 1990; CSC, 1989; Gilstrap, 1990).

Briefly, ESDM is a risk-driven development methodology. Areas of risk in the development are identified, and work on any one phase is directed toward reducing the next highest remaining risk. Risks due to uncertainty are inherent in any system development but are greater at the outset of expert system development because of the unknowns in the modeling of the human expert's cognitive behavior.

The system life cycle in ESDM is divided into five stages, each of which is further divided into five steps as shown in the spiral model in Figure 1. This spiral model is similar to that recommended by Boehm (1986) for conventional systems. The Boehm spiral model is also used for expert systems by Stachowitz and Combs (1987), Stachowitz et al. (1988), O'Keefe and Lee (1990), and others.

At the time the ESDM task was initiated, it was decided that the establishment of procedures and methods for performing validation and verification (V&V) would be deferred. The original focus was on methodology; however, V&V are concerned with both methodological and technical issues. Because of the criticality of NASA missions, it is necessary to be able to certify the behavior of expert systems and to verify their performance. A study, which is the basis for this paper, was undertaken to determine the steps necessary to integrate V&V into ESDM.

As pointed out by Green and Keyes (1987), V&V has seldom been done for expert systems. The primary reason is that requirements analyses are not always done for expert systems. Without requirements, it is, by definition, not possible to perform V&V. A secondary reason is that there are special problems, discussed in more detail later, in doing V&V for expert systems, even when requirements exist.

ESDM recommends that requirements be prepared in all expert system developments, but does not mandate them. However, requirements should not be

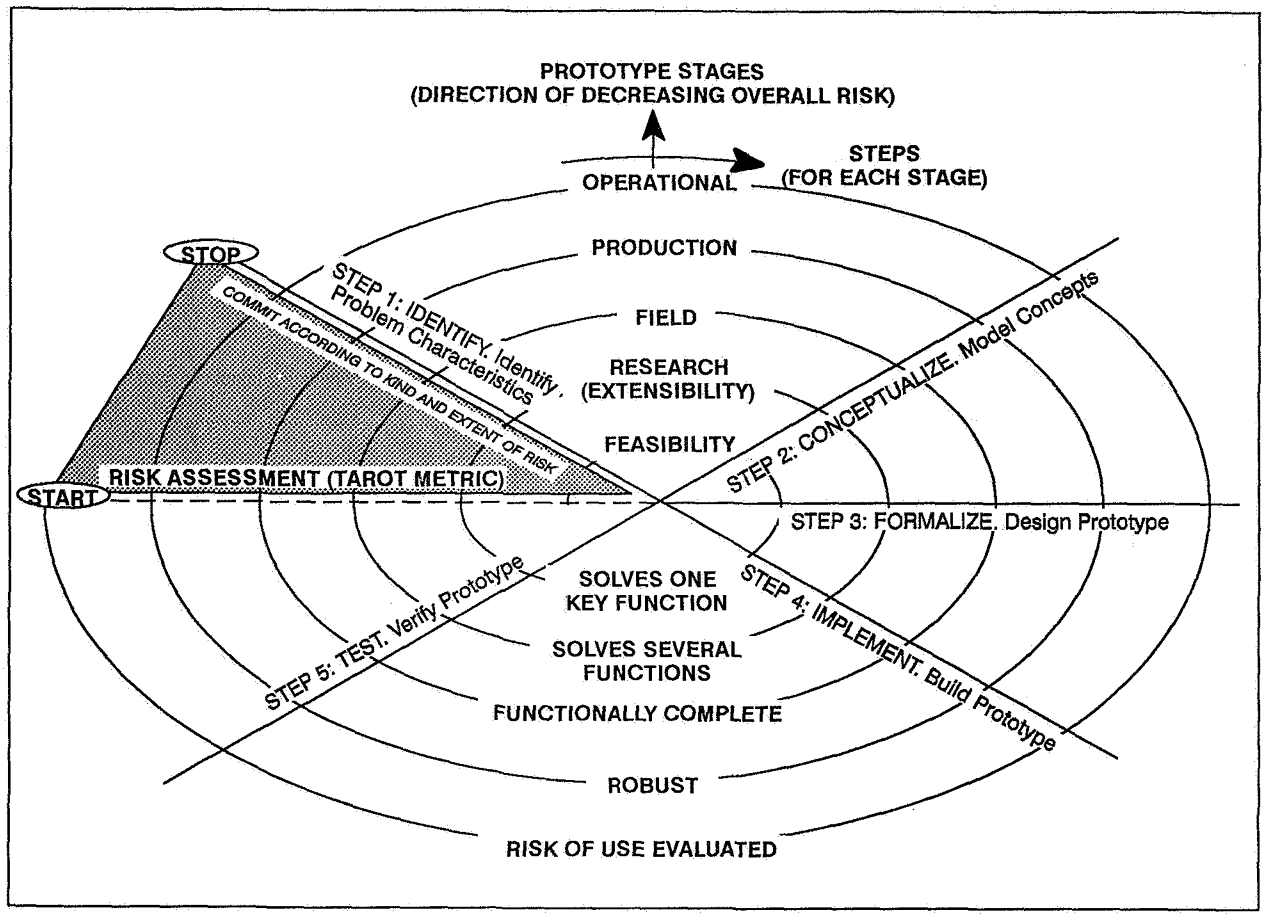

PROTOTYPE STAGES
(DIRECTION OF DECREASING OVERALL RISK)
STEPS
(FOR EACH STAGE)
OPERATIONAL
PRODUCTION
FIELD
RESEARCH
(EXTENSIBILITY)
FEASIBILITY
STOP
START
COMMIT ACCORDING TO KIND AND EXTENT OF RISK
RISK ASSESSMENT (TAROT METRIC)
STEP 1: IDENTIFY. Identify Problem Characteristics
STEP 2: CONCEPTUALIZE. Model Concepts
STEP 3: FORMALIZE. Design Prototype
STEP 4: IMPLEMENT. Build Prototype
STEP 5: TEST. Verify Prototype
SOLVES ONE
KEY FUNCTION
SOLVES SEVERAL
FUNCTIONS
FUNCTIONALLY COMPLETE
ROBUST
RISK OF USE EVALUATED

**Figure 1. Spiral Model of ESDM**

prepared until the major uncertainties about the system have been resolved. Typically, this does not occur until after the field stage of work has been completed.

## Validation and Verification

The terms *validation*, *verification*, and *testing* are defined as follows in the Institute for Computer Sciences and Technology (ICST) special publication on V,V&T (1988):

> "*Validation, Verification and Testing* (V,V&T): A process of review, analysis, and testing employed throughout the software development lifecycle...which helps ensure the production of quality software.
>
> "*Validation*: The determination of the correctness of the end product (code) with respect to the software requirements, i.e., does the output conform with what is required?
>
> "*Verification*: The determination that each phase and subphase of the development lifecycle is correct, complete, and consistent with itself and with its predecessor product.
>
> "*Testing*: The examination of program behavior by executing the program on sample or operational data sets to determine the correctness of the program."

Verification is more concerned with the structure or form of a system, while validation is more concerned with behavior. In simple terms, as Boehm (1984) expresses it, in verification we ask

> "Am I building the product right?"

and in validation we ask

> "Am I building the right product?"

Green and Keyes (1987) point out that verification is, in many cases, a "paper" activity, while validation is a "live" activity involving testing. In conventional systems, validation and verification tend to find different types of errors; they are complementary processes, and neither is sufficient by itself to uncover all the errors in a system.

## Testing and Inspection

The primary tools used in V&V are testing and inspection. The purpose of testing is to find errors, omissions, or unnecessary elements in a system. The most common types of tests used in conventional development are

- Unit tests—these are primarily path tests and are directed toward finding gross structural errors
- Integration tests—these are performed each time a new module is added to a system being developed
- System tests—these are performed to ensure that a system as constructed meets the requirements specified for the system
- Acceptance tests—these tests are performed by the customer or user on delivery of a system

Unit and integration tests help to verify that the system has been built correctly. System tests validate the system with respect to requirements. The validation of requirements (against the real world) may require special tests to be designed and executed. For example, in some conventional system development, field tests under actual conditions might be required to validate the system requirements. Such tests would be in addition to system tests for ensuring that the system satisfies requirements.

In addition to testing, software system inspection is also performed to uncover errors or to verify system consistency or correctness. Inspection techniques have also been used in developing expert systems, and a wide variety of special inspection techniques have been designed for expert systems. For example, the human expert is given the knowledge base for an expert system and asked to review it for errors of omission or commission. A number of these methods are listed and briefly described in the ESDM user's manual (CSCa, 1988).

## Provisions for Expert System V&V

Based on literature review and analysis of the problem, a methodology must provide for the following in order to support V&V:

- Requirements generation (O'Keefe & Lee, 1990; Culbert et al.[a], 1987; Culbert et al.[b], 1987; Barrett, 1990; Preece, 1990)
- V&V planning (Barrett, 1990), defining the following:
  - Objective of each test
  - Criteria for each test
  - Data required to measure attainment of objective
  - Data acquisition procedures for collecting required data
  - Analytical procedures for determining whether criteria have been met
- Guidelines (Green & Keyes, 1987) for the following:
  - Procedures for tests and inspections
  - Requirements-traceable tests
  - Parameters of system to be tested
- Special testing of knowledge-based systems, as required
- Automatic testing tools (Stachowitz & Combs, 1987; Stachowitz et al., 1988; Gupta & Biegel, 1990) to support the following:
  - Test requirements determination and documentation
  - Test planning
  - Execution of test and inspection procedures
  - Execution of special testing
  - Requirements tracing

Without requirements, V&V are simply not possible in the usual sense of the term. The cost of preparing requirements in some later stage of expert system development, after sufficient information has been acquired, is considered a good investment. Requirements provide the basis for testing all subsequent prototypes and are essential for testing critical applications in operational environments.

The methodology for developing expert systems must mandate test planning and provide for independent testing and inspection. It must also provide guidelines and specify procedures for performing tests and inspections. Experience has shown that for best results, testing and implementation of expert systems should be independently managed, just as for conventional systems development.

Expert systems are not conventional systems, however, and we can test expert systems in ways that are impossible with conventional systems. The special kinds of testing that are possible for expert systems exist because of the knowledge base, the different types of knowledge representation, and the non-procedural methods of reasoning that may be used in expert systems. Examples of special tests include searching for contradictions in the knowledge base, ensuring that explanations given by the system are understandable and reasonable, and ensuring that the system does not produce patently absurd or illogical conclusions. Details of such tests may depend strongly on the specific knowledge representation used in the system and require more, perhaps much more, preparation than do tests to ensure adherence to system requirements.

Most expert systems developers would agree that it is desirable to have automated tools for testing, and some would argue that it is essential. Stachowitz and Combs (1987) state:

> "It is our conjecture that software validation can be more easily performed in a knowledge-based system environment. In such an environment the number of life cycle steps is reduced from the traditional four (requirements development, specification development, design development, code development) to just the first two, resulting in a considerable reduction of the amount of validation work to be performed."

Also, Gupta and Biegel (1990) list seven limitations of manual test planning and execution in support of expert system testing, ranging from possible lack of objectivity to the "astronomically large" number of test cases needed to extensively test a system. Certainly the likelihood is quite high that any complex expert system designed for highly critical requirements will require special tests that are not available from generic products. An expert system development methodology must make provisions for evaluating criticality and for developing special test procedures in parallel with the system, particularly in the high-criticality cases.

## Special Expert System V&V Tests

Special tests, for which there may be no generic, commercially available testing tool, are conducted to resolve differences among multiple experts and to verify the following:

- Correctness of reasoning
- Inference engine
- Knowledge base
- Correctness and value of the output advice/ actions
- Correctness of explanations
- Expertise of the expert
- System boundaries

In addition to validating or verifying the knowledge base, performance of one or more of the above tests may be necessary before the system can be certified for critical applications. Test design should be considered to be just as important as the design of the knowledge representation scheme, the inference engine design, or the knowledge engineering methods used on the project.

Culbert and colleagues discuss the issues involved in performing many of these special kinds of tests in the context of NASA systems (a, 1987) and present their own expert system development methodology (b, 1987), one that supports verification and validation. The approach they describe makes use of panels of domain experts, users, and managers with system responsibilities to ensure that all applicable viewpoints are represented, both during development work and during inspections (Culbert et al.[b], 1987). The life cycle used in this approach consists of four phases:

- Problem definition
- Initial prototype
- Expanded prototype
- Delivery/maintenance

The emphasis in this approach is not on the use of tests, automated or otherwise; rather, it is on ensuring that all relevant human skills are brought to bear in the development process—an important objective, and one that should be supported.

The mechanism used for verifying the correctness of knowledge bases depends on the kind of reasoning process used in the system. The framework for verifying knowledge bases that are developed for use with ordinary logic is readily available. For example, resolution can always be used to mechanize the determination of rule-base consistency, although it is not the most efficient way. However, knowledge bases developed for nonmonotonic reasoning require an extension of the procedures developed for monotonic reasoning. Chang and colleagues (1990) discuss this problem and indicate how the automated testing tool, Expert Systems Validation Associate (EVA), could be extended to the nonmonotonic case.

The long-range goal of EVA was to construct an integrated set of generic tools to validate any knowledge-based system in any expert system shell (Stachowitz et al., 1988). It currently provides 11 tests that assist in verifying knowledge-based systems, ranging from structure and logic checks to test case generation and behavior verification. The addition of a capability for handling nonmonotonic logic would be a significant enhancement to the original system.

The problems of validating an expert or of resolving differences among multiple experts are of a very different order from the problems of verifying an expert system or features of an expert system. An evaluation technique called the *analytic hierarchy process (AHP)*, developed by Saaty (1980), was adapted by Liebowitz (1985) for evaluation of expert systems. The AHP technique makes use of pairwise subjective evaluations (Is A more important than B?) to achieve an integrated, global evaluation and ranking of many factors. A key feature of the method is that it is tolerant of intransitivity of relations (team A beats team B, which beats team C, which then beats team A). This technique can also be used to resolve the differences among multiple experts.

## ESDM Modifications

The GSFC Expert System Development Methodology (ESDM), represented by the spiral model of Figure 1, currently supports some of the provisions of a methodology needed for V&V. The following lists the changes to ESDM needed to more fully support V&V:

- *Requirements generation*. ESDM does not now mandate the preparation of requirements. ESDM will be modified to make requirements mandatory in all high-criticality projects to ensure that V&V are possible.
- *V&V planning*. ESDM currently requires the design of tests to check for stage completion. ESDM will be modified to specify V&V for all systems for which requirements are developed. ESDM documentation requirements will also be modified to make the test plan a formal project document.
- *Guidelines for procedures, tests, test parameters*. ESDM will be modified to provide guidelines for tests and testing procedures that can be used. The choice of tests for a particular project will be left to the project manager and the knowledge engineer.
- *Special testing*. ESDM will provide a checklist of special tests. This checklist will include the type of knowledge representation, the type of inferencing method, and the language or shell that is appropriate for each special test.
- *Automatic testing tools*. For critical projects, ESDM will specify an analysis of automated tool requirements. ESDM will also provide guidelines for estimating the costs, labor, and schedule to develop such automated tools in parallel with the expert system.

The above are the primary changes to ESDM needed to support V&V. ESDM will not mandate the specific tests that must be performed, but will provide guidelines that can be used by project managers and developers. In the future, the guidelines in ESDM are expected to be replaced by standards. At present, there is not sufficient experience with V&V in any expert system methodology to be able to define such standards. To force standards at this early stage might unduly restrict the technology of knowledge-based systems.

## Evaluation of Expert Systems

Testing is a special case of evaluation of systems. As discussed by Liebowitz (1985), evaluations are helpful in determining whether an expert system is meeting its intended goals. The main difference between evaluation and testing is that evaluations are performed with respect to a specific set of objectives or purposes, whereas tests are evaluated with regard to correctness or performance. Evaluation usually requires the use of subjective judgment with respect to the objectives or purposes of the system:

> "In your judgment, is system X suitable for application (or purpose) Y?"

The procedure for making evaluations is more elaborate than is implied by the above question. Usually, the features or characteristics of the system are evaluated one at a time, and an overall evaluation is obtained by integrating or producing a weighted average of the estimates of suitability over all the features. In V&V, the objectives or purposes are specified by the requirements of the system, and subjective judgment is reduced to a pass/fail decision. The overall global decision to accept or reject is usually based on the percentage of cases that pass. For expert systems, it is not reasonable to require 100-percent success for all tests, because a human expert does not succeed 100 percent of the time.

The special tests that can be constructed for expert systems correspond to a set of pass/fail tests for the properties, characteristics, or features of expert systems. For evaluations of all sorts, it is recommended that the features selected for evaluation be both independent and mutually exhaustive of all relevant characteristics (i.e., they should cover all of the important properties of the system). A checklist of such features relevant to the evaluation of a potential expert system for suitability and worth was included in ESDM for use with the Test for Application of Risk-Oriented Technology (TAROT).

A number of authors have developed lists of expert system features, including Liebowitz (1985), Marcot (1987), Boehm et al. (1978), and Stachowitz and Combs (1987). The following list of system characteristics was obtained by consolidating and then eliminating duplications from these other lists. It is more general than the current ESDM list of system characteristics in that it covers more factors than those needed for evaluation of project suitability. It will replace the current ESDM list.

- Structural parameters
  - Completeness
  - Comprehensibility
  - Conciseness
  - Correctness (freedom from contradiction)
  - Consistency (freedom from anomaly)
  - Legibility
  - Modularity
  - Self-descriptiveness
  - Simplicity
  - Structuredness
  - Understandability
- Behavioral parameters
  - Utility
    - Accuracy
    - Effectiveness
    - Correctness (correctness of output)
    - Credibility
    - Intelligibility
    - Ease of use
    - Producibility
    - Speed
  - Operability
    - Efficiency
    - Flexibility
    - Interoperability
    - Maintainability
    - Portability
    - Reliability
    - Reusability
    - Robustness
    - Sensitivity
    - Stability
  - Quality
    - Completeness (breadth, depth)
    - Conciseness
    - Consistency
    - Correctness
    - Integrity
    - Maintainability
    - Reliability
    - Testability
  - Suitability
    - Worth
    - Risk
    - Benefits
    - Costs
    - Urgency/priority
    - Criticality

The special tests needed for a given expert system project are all related to one or more of the structural and behavioral characteristics listed above. System developers can use the checklist to help select those features that are critical in their application. Once selected, special tests must then be developed to examine each of the features or characteristics.

## Summary

Although V&V are difficult to accomplish for expert systems, the main problem has been that requirements have, in many cases, been missing. Without requirements, V&V are not possible. However, even when requirements are available, performing V&V on expert systems may not be easy because of the large number of different types of tests and inspections that are possible for such systems.

ESDM, the GSFC methodology for development of expert systems, is being upgraded to provide guidelines and checklists that assist managers and developers in planning, documenting, and performing V&V tests and inspections to certify expert systems for use in critical applications.

There is no one, simple, magic solution to the achievement of reliable, bug-free, certifiable software of any kind. High-quality, reliable software requires many different types of tests and inspections, as well as adherence to sound design and coding practices. The best way to achieve quality expert systems is for project managers and developers to be aware of the options available for testing and inspection and to plan on using the appropriate tools at the right time. ESDM modifications support these objectives.

## References

Barrett, B. W. (July 1990). A software quality specification methodology for knowledge-based systems, *Workshop Notes* for AAAI-90 Workshop on Knowledge-Based Systems Verification, Validation, and Testing, Boston, MA.

Boehm, B. W. (January 1984). Verifying and validating software requirements and design specifications. *IEEE Software Journal.*

Boehm, B. W. (March 1986). A spiral model of software development and enhancement. *ACM Software Engineering Notes.*

Boehm, J., Brown, R., Kaspar, H., Lipow, M., MacLeod, G. J., & Merrit, H. H. J. (1978). *Characteristics of software quality,* North-Holland.

Chang, C. L., Stachowitz, R. A., & Combs, J. B. (July 1990). Validation of nonmonotonic knowledge-based systems. *Workshop Notes* for AAAI-90 Workshop on Knowledge-Based Systems Verification, Validation, and Testing, Boston, MA.

Computer Sciences Corporation (CSC) (a) (September 1988). Expert system development methodology (user's guide), DSTL 90-004.

CSC (b) (September 1988). Expert system development methodology (policy document), DSTL-90-005.

CSC (c) (September 1988). Expert system development methodology (reference manual), DSTL-90-006.

CSC (March 1989). Expert system development methodology; framework for evaluation of ESDM.

Culbert, C., Riley, G., & Savely, R. T. (August 1987). Approaches to the verification of rule-based expert systems. *Proceedings of SOAR'87: Space Operations—Automation and Robotics Conference*, Houston, TX.

Culbert, C., Riley, G., & Savely, R. T. (September 1987). An expert system development methodology which supports verification and validation. *Proceedings of Fourth IEEE Conference on Artificial Intelligence Applications*, NASA/Lyndon B. Johnson Space Center, Houston, TX.

Gilstrap, L. (May 1990). Evaluation of a proposed expert system development methodology: Two case studies. *1990 Goddard Conference on Space Applications of Artificial Intelligence.*

Green, C. J. R., & Keyes, M. M. (1987). Verification and validation of expert systems. *Proceedings: WESTEX-87—Western Conference on Expert Systems*, IEEE, pp. 38-43.

Gupta, U. C., & Biegel, J. (July 1990). RITCaG: A rule-based intelligent test case generator. *Workshop Notes* for AAAI-90 Workshop on Knowledge-Based Systems Verification, Validation, and Testing, Boston, MA.

Institute for Computer Sciences and Technology (ICST) (1988). *Software validation, verification, testing, and documentation*, S. Andriole (Ed.), special publication on V,V&T, Petrocelli Books, NJ, p. 7.

Liebowitz, J. (December 1985). Evaluation of expert systems: An approach and case study. The Second Conference on Artificial Intelligence Applications, Miami Beach, FL.

Marcot, B. (August 1987). Testing your knowledge base. *AI Expert*, pp. 42-47.

O'Keefe, R. M., & Lee, S. (July 1990). An integrative model of expert system verification and validation. *Workshop Notes* for AAAI-90 Workshop on Knowledge-Based Systems Verification, Validation, and Testing, Boston, MA.

Preece, A. D. (July 1990). The role of specification in expert system evaluation. *Workshop Notes* for AAAI-90 Workshop on Knowledge-Based Systems Verification, Validation, and Testing, Boston, MA.

Saaty, T. (1980). *The analytical hierarchy process*, McGraw-Hill, NY.

Sary, C., Gilstrap, L., & Hull, L. G. (May 1990). Expert system development methodology (ESDM). *Proceedings of Fifth Conference on AI for Space Applications*, Huntsville, AL.

Stachowitz, R. A., & Combs, J. B. (1987, January 6–9). Validation of expert systems. *Proceedings of Hawaii International Conference on Systems Sciences*, Kona, HI.

Stachowitz, R. A., Combs, J. B., & Chang, C. L. (1988). Validation of knowledge-based systems. In E. Heer, H. Lum (Eds.), *Machine intelligence and autonomy for expert systems*, AIAA, Washington, DC, pp. 125–142.

N91-22788

# Techniques and Implementation of the Embedded Rule-Based Expert System Using Ada.

Eugene M. Liberman
Sverdrup Technology Corporation
2001 Aerospace Parkway
Brook Park, Ohio 44142

and

Robert E. Jones
National Aeronautics and Space Administration
Lewis Research Center
Cleveland, Ohio 44135

## ABSTRACT

Ada is becoming an increasingly popular programming language for large Government-funded software projects. Ada with its portability, transportability, and maintainability lends itself well to today's complex programming environment. In addition, expert systems have also assumed a growing role in providing human-like reasoning capability and expertise for computer systems.

This paper discusses the integration of expert system technology with Ada programming language, specifically a rule-based expert system using an ART-Ada (Automated Reasoning Tool for Ada) system shell. The Inference Corporation developed ART-Ada under a program sponsored by the NASA Johnson Space Center. The NASA Lewis Research Center was chosen as a beta test site for ART-Ada. The test was conducted by implementing the existing Autonomous Power EXpert System (APEX), a Lisp-based power expert system, in ART-Ada.

Three components, the rule-based expert system, a graphics user interface, and communications software make up SMART-Ada (Systems fault Management with ART-Ada). The rules were written in the ART-Ada development environment and converted to Ada source code. The graphics interface was developed with the Transportable Application Environment (TAE) Plus, which generates Ada source code to control graphics images. SMART-Ada communicates with a remote host to obtain either simulated or real data. The Ada source code generated with ART-Ada, TAE Plus, and communications code was incorporated into an Ada expert system that reads the data from a power distribution test bed, applies the rules to determine a fault, if one exists, and graphically displays it on the screen.

The main objective of this study, to

conduct a beta test on the ART-Ada rule-based expert system shell, was achieved. The system is operational. New Ada tools will assist in future successful projects. ART-Ada is one such tool and is a viable alternative to the straight Ada code when an application requires a rule-based or knowledge-based approach.

Keywords: Ada, Expert System, Rule-Based, Knowledge-Based.

## INTRODUCTION

Ada is becoming the language of choice for large Government-funded projects as increasing software size and complexity drives the cost of development and maintenance upward. Ada (J.G.P Barnes, 1984) language offers standardization, portability, maintainability, and readability. All these features provide an excellent environment for developing complex software, increasing programmers' productivity, and encouraging team work.

Many large-scale software projects targeted toward space applications involve health monitoring and control of mission-critical systems. The necessary expertise to maintain reliable space mission operations is often unavailable due to limited resources. Even if the expertise is available, routine system health monitoring remains a time-consuming and tedious task. Expert systems have been taking an increasingly larger role in providing knowledge for problem resolution and in handling tedious tasks.

The integration of expert system technology with Ada programming language results in a powerful combination for use in many large-scale computer applications. An expert system shell that generates Ada code is one such combination. ART-Ada is an expert system shell that generates Ada code.

This paper describes the techniques for and implementation of a rule-based expert system using the ART-Ada expert system shell. The Inference Corporation developed ART-Ada under a program sponsored by the NASA Johnson Space Center. The NASA Lewis Research Center was chosen as a beta test site for ART-Ada. The test was conducted by implementing the existing Autonomous Power EXpert system (APEX) (Ringer and Quinn, 1990) with ART-Ada. APEX is a Lisp-based power expert system designed as a fault diagnostic adviser for the monitoring and control of 20 kHz power distribution test bed. APEX is being developed at the NASA Lewis Research Center.

## SYSTEM DESCRIPTION

The system description is limited to a discussion of the software implementation of SMART-Ada and the hardware on which the software was implemented. SMART-Ada is written on a SUN SPARCstation 1 equipped with 16 megabytes of memory and approximately 0.5 gigabyte of hard disk capacity.

The operating system is SUN OS 4.0.3c running X Window X11R3. The following software packages were used to implement SMART-Ada:

(1) ART-Ada
(2) Transportable Application Environment (TAE) Plus
(3) Remote procedure call protocol (RPC)
(4) X Window system
(5) X Window Ada bindings
(6) VERDIX Ada compiler

ART-Ada is an expert system shell for

the development of rule-based or knowledge-based expert systems. ART-Ada is based on the Automated Reasoning Tool for Information Management (ART-IM). One important feature of ART-Ada is its ability to generate Ada source code from a knowledge-based or rule-based application written in ART-IM. The syntax of ART-Ada is compatible with ART-IM, making code developed in ART-IM transportable to ART-Ada as long as no machine-dependent features of either software are used.

TAE Plus is a software package developed by the NASA Goddard Space Flight Center. The package is an integrated environment for creating and running window-based applications with a graphical point-and-click user interface. TAE Plus is based on the X Window System. TAE Plus was chosen for the SMART-Ada graphical user interface because it can generate Ada code.

RPC is a communications protocol developed by Sun Microsystems. This protocol allows machines of different types to interact with each other on a procedure level. This interaction means that one machine can call a procedure on the other, pass arguments to the procedure and then receive any returned values. RPC software for SPARCstation 1 was developed in the C programming language. An Ada-to-C interface is required to allow SMART-Ada software to take full advantage of RPC protocol. This interface was written for this project.

The X Window System, developed at MIT, has become the industry standard and runs on a wide range of computing and graphics machines. The X Window System provides a powerful windowing environment for producing high-performance graphical interface. Since the X Window System was developed in C programming language, an interface between the Ada code and the X Window C code is necessary to take advantage of the X Window System features.

Ada language bindings to the X Window System are a package developed by Science Applications International Corporation (SAIC). The bindings provide the necessary interface between X Window C libraries and the Ada code in SMART-Ada. The use of bindings allows the Ada code to take full advantage of the powerful X Window System procedures.

The VERDIX compiler is used for compilation and executable code generation. The VERDIX compiler was recommended by the Inference Corporation for development and implementation of SMART-Ada. The VERDIX compiler also allows for a good Ada-to-C interface, which is an important feature needed for accessing X Window System procedures with the Ada code.

## SMART-Ada system components

The SMART-Ada system consists of three major components:

(1) A rule-based expert system
(2) A graphics user interface (GUI)
(3) Communications software

These three components were integrated to make up the entire SMART-Ada system. All components are implemented in Ada programming language. However, in two cases the interface between C libraries and the Ada code is used to enable Ada to access existing C software libraries. The interface between X Window procedures and the Ada graphics programs was implemented with the SAIC Ada language bindings to the X Window System.

The interface between RPC software and Ada was a part of SMART-Ada development. Each component and its implementation is discussed in greater detail in the following paragraphs.

## Rule-Based expert system

The knowledge-based rule set was developed with ART-IM because of its user-friendly development environment. ART-IM's powerful debugging features were heavily utilized.

The rule-based expert system is responsible for monitoring the operational state of a 20 kHz power distribution test bed. If an abnormality occurs in the test bed, the expert system detects the fault condition and isolates the probable cause. It performs fault detection by comparing measured operating values to the expected values, accessing information and rules contained within its knowledge base in order to isolate the fault.

A collection of rules in the knowledge base forms groups of logical subtasks. The logical subtasks are connectivity, initialization, detection, isolation, affected loads, and recommended actions. The connectivity subtask determines the power distribution configuration of the 20 kHz power distribution test bed. The initialization subtask calculates expected values for voltages, currents, and power on the basis of the test bed configuration and the physical properties of the test bed components. The detection subtask compares the expected values obtained in the initialization subtask with the actual system values obtained from the test bed. The isolation subtask isolates the problem and assists the expert system in determining where in the circuit problems have occurred. The affected loads subtask determines which load is affected by the faults in the system. The recommended action subtask, in the future development of the system, will recommend a new configuration for the power distribution or will automatically reconfigure the power distribution to alleviate existing problems.

The order of rule execution in a subtask is determined by the salience of each rule. The higher salience rules execute first and the lower salience rules execute last. The last rule in the subtask contains the lowest salience so that it will execute only after all the rules in the subtask have executed. In SMART-Ada, the lowest salience rule in each subtask is used to "clean up" the current subtask and set the environment for the next subtask. Since SMART-Ada is a monitoring system, the rule-based expert system must execute continuously. The last subtask sets the environment for the first subtask, creating the effect of an infinite loop.

## Expert system and Ada Interface

In order to execute Ada language subprograms from ART-Ada, an interface between ART-Ada and Ada language is required. The interface is accomplished by defining an Ada USER package (Figure 1). Within ART-Ada, USER is a reserved symbol for accessing external Ada code from ART-Ada rules. Parameter passing is also possible between ART-Ada and subprograms in the USER package. The parameters, however, have to adhere to the syntax of ART-Ada. SMART-Ada uses subprograms contained in the USER package to control the graphical user interface and to obtain data from the 20 kHz power distribution test bed.

ART-Ada contains a feature that allows one function to be specified as asynchronous. A function defined as asynchronous is automatically invoked before and after each rule execution. The subprogram named ASYNCH_FUN in the USER package is defined as an asynchronous function. ASYNCH_FUN is responsible for reading data from the test bed or using simulated data and providing data to the rule-based expert system (Figure 2) as well as to the graphics interface. All subprograms in the USER package are capable of accessing and modifying data in the ART-Ada code.

As the rules are being executed, the asynchronous function waits until the last rule has been completed. The last rule execution is an indicator for ASYNCH_FUN to read new data from the 20 kHz power distribution test bed and introduce the new data to the expert system for evaluation. The new data are also dispatched to the graphics interface for a system status update. The execution of the expert system then continues with execution of the first subtask.

The WRITE_TO_FILE subprogram in the USER package is responsible for writing the explanations of errors that occur as a result of the rule-based expert system execution. The rule-based expert system provides the file name and the line-by-line text that is to be entered in the file specified in the file name parameter. The file name is determined by which rule subtask is executing at the moment. The detection rule subtask, for example, writes to the DETECT.DAT file, and the isolation rule subtask writes to the ISOLATE.DAT file. The text that goes in a file is determined by the executing rule. The FLAG_ERROR subprogram is invoked when an error condition is present. The rule that invokes the subprogram must provide the switch name and the faulty component to the subprogram. The code in the FLAG_ERROR subprogram communicates by means of the graphics interface and sets the visual alert in the graphics module.

## Graphics User Interface

The graphics user interface (GUI) for the SMART-Ada was developed with TAE Plus. TAE Plus was chosen for this application because of its capability to generate Ada code for the developed graphics.

The DDO (data driven object) feature of TAE Plus is used to achieve the dynamic display of the system status. The screens and the DDO's were created with the TAE Plus graphics editor. The screens show the system status at high and low levels. The high- level screen shows the overall system state and configuration. The faults with the system components are shown with appropriate color and a warning banner. The lower-level screens provide a more detailed look of a selected component. The currents, voltages, and power indicators contain the up-to-date values. The faults with any of the values are also marked with appropriate colors and a warning banner.

An explanation for the errors is also provided. The Rule-Based expert system generates the text corresponding to the system status. The text is stored in the file set up for that purpose. The file DETECT.DAT contains the result of the detection subtask execution, the file ISOLATE.DAT contains the results of the isolation subtask execution, etc. The GUI displays the contents of these files to provide the user with the explanation of the system status.

### Communications Software

SMART-Ada communicates with a remote host to obtain either simulated or real data. The communications link between SMART-Ada and the data acquisition software on the remote host is accomplished by using the transmission control protocol / internet protocol (TCP/IP) and RPC protocol. The remote host must contain the procedure that is invoked to provide data to SMART-Ada. The remote procedure returns system status data that are interpreted by the system.

## BETA TEST RESULTS

The main objective of SMART-Ada implementation was to conduct a beta test on the ART-Ada rule-based expert system shell. The beta test revealed that it is, indeed, possible to implement an Ada-based application by using the ART-Ada expert system shell. However, a few problems were found as the project developed. The first problem occurred in an attempt to deploy a set of rules. The set of rules was executing properly in the development environment, but when the rules were converted to Ada source code and an executable code was generated, the program failed to run. A constraint Ada error was raised when execution was attempted. The problem was reported to the Inference Corporation who acknowledged and fixed the problem and sent us corrected version of ART-Ada. The second problem was found with the ART-Ada package. A variable contained no value after a value had been assigned to it. The Inference Corporation has acknowledged the problem and promised to fix the package for the next version of the software and has suggested a way to work around the problem.

The maintainability issues of the ART-Ada code must be addressed. Unfortunately, the tendency is to address maintainability of the Ada code generated from the ART-Ada application. The proper way to maintain the ART-Ada application is through the ART-Ada code itself. System maintenance will become much easier if emphasis is placed on ART-Ada code maintenance rather than on Ada code maintenance.

## CONCLUDING REMARKS

As Ada programming language becomes more and more prevalent, new Ada tools will assist in making Ada projects even more successful. ART-Ada is the first rule-based tool available for Ada programming language. The ART-Ada package has its problems, like any initial version of a major software release. But the authors feel that ART-Ada is a viable alternative to the straight Ada code for an application that requires a rule-based or knowledge-based approach.

```
with ART;
package USER is

    function ASYNCH_FUN;

    procedureWRITE_TO_FILE
        (NAME_OF_FILE   : in  ART.ART_OBJECT;
         STRING_VALUE   : in  ART.ART_OBJECT);

    procedure FLAG_ERROR
        (SWITCH_NAME    : in  ART.ART_OBJECT;
         COMPONENT      : in  ART.ART_OBJECT);

    end USER;
```

Figure 1. Ada USER package

```
with USER; use USER;
(def-user-fun asynch-fun
                    :args ()
                    :returns            :void
                    :compiler           :verdix)

(def-user-fun write-to-file
                    :args (
                    (FILE-NAME          :ART-OBJECT)
                    (STRING_VALUE       :ART-OBJECT))
                    :returns            :void
                    :compiler           :verdix)

(def-user-fun flag-error
                    :args (
                    (SWITCH :ART-OBJECT)
                    (COMPONENT          :ART-OBJECT))
                    :returns            :void
                    :compiler           :verdix)

                    .
                    .
                    .
                    (set-asynch-func async-func)
```

Figure 2. The rule-based expert system use of USER package

## ACKNOWLEDGMENTS

The authors would like to thank Todd Quinn of Sverdrup Technology Corporation for his invaluable technical and editorial assistance and the entire Space Electronics Division AI laboratory staff for their comments and support.

## REFERENCES

Barnes, J.G.P (1984). Programming in Ada. 2nd ed., Reading, MA: Addison-Wesley.

Ringer, M.J. and Quinn, T.M. (1990), Autonomous Power Expert System. NASA CR-185263.

N91-22789

# A Reusable Knowledge Acquisition Shell - KASH

*Christopher Westphal*
*Stephen Williams*
*Virginia Keech*

SYSCON Corporation
1000 Thomas Jefferson St., N.W.
Washington, D.C. 20007

## Abstract

*KASH (Knowledge Acquisition SHell) is proposed to assist a knowledge engineer by providing a set of utilities for constructing knowledge acquisition sessions based on interviewing techniques. The information elicited from domain experts during the sessions will be guided by a question dependency graph (QDG). The QDG, defined by the knowledge engineer, will consist of a series of control questions about the domain that are used to organize the knowledge of an expert. The content information supplied by the expert, in response to the questions, will be represented in the form of a concept map. These maps can be constructed in a top-down or bottom-up manner by the QDG and used by KASH to generate the rules for a large class of expert system domains. Additionally, the concept maps can support the representation of temporal knowledge. The high degree of reusability encountered in the QDG and concept maps in KASH can vastly reduce the development times and costs associated with producing intelligent decision aids, training programs, and process control functions.*

## Introduction

The field of expert systems has claimed many successful applications ranging from simple tasks, such as monitoring a valve or fluid rate, to very complex tasks that may include process scheduling or design. However, to construct a system requires the developer to isolate a set of rules that will guide the decision making process. Rules are conventionally defined during a series of interviews between a domain expert and a knowledge engineer. The encoding and representation of the extracted domain knowledge have proven to be difficult barriers to overcome and have been commonly deemed the *knowledge acquisition bottleneck.* To reduce the amount of time required for this stage of expert systems development, knowledge acquisition techniques have been extended beyond the traditional verbal interview methods to include both semi-automated tools that improve knowledge engineering efficiency and fully autonomous approaches (machine learning) that attempt to infer knowledge directly from examples of expert behavior.

To date, numerous tools have been developed to address the many aspects of knowledge acquisition. The tools tend to be highly user oriented with the interfaces and structure of information significantly well advanced to accommodate a rapid and easy facilitation of knowledge. The application domains of these tools can be aligned into two categories; *analysis* (e.g., diagnosis, identification, interpretation) and *synthesis* (e.g., scheduling, planning, design). Analysis is a top-down process providing an examination of a set of goals that are decomposed into simpler and more basic elements and relationships. For example, FIX [Rodi, Pierce and Dalton, 1989] is a diagnostic system based on class attributes for representing fault detection and isolation. Synthesis is a bottom-up process supporting the composition or combination of given facts and observations that are used to construct a set of goals. For example, SALT [Marcus, 1988] is a tool based on a propose-and-revise method for designing elevator configurations.

The above classes of tools work well within their intended domains, but are not easily transferred to other applications. Consequently, the time and effort required for producing the systems must be reinvested for each instance. This will increase the costs to the developers and subsequently decrease reusability and portability of the tools across domain applications. To address the different categories (analysis or synthesis), a system must be able to work within the constraints set by each. Expert system shells (e.g., KEE, ART, CLIPS, NEXPERT) currently provide the capacity to build top-down or bottom-up applications by supplying the developer with a finite collection of objects to represent the domains. The manner in which the objects are defined and related to other objects determines the degree of acceptability of the system. Therefore, if the focus of producing a knowledge acquisition system is placed on the *structure* rather than on the *content* of knowledge, a more diverse selection of applications may be addressed.

Yet, much of the previous research with knowledge acquisition tools has been confined to specialized areas of interest with minimal attention allotted to the dissemination of the information. Using the control knowledge represented in the structure of information will facilitate the reuse of knowledge between applications, and the context information can then be elicited for the structures from the experts for each unique application. Slight variances in the control structures can radically affect the content information obtained. Multiple expert based systems are naturally one of the primary benefits of this type of approach. Thus, the domains of interest can extend across a large number of applications including training systems, mission planning, and manufacturing. In doing so, the standardization of knowledge and the issues of reusability become a reality.

### *Reusing Knowledge Structures*

The reusability issues of knowledge based systems have slowly moved into mainstream considerations of expert system developers. Many government, industry, and academic sectors have participated in the development of expert systems technologies and have produced a considerable set of disheveled applications. The information obtained from the knowledge engineering phase, much of which is repeated between applications, is typically the largest cost of expert systems production [McGraw, 1989]. If a reusable data store of reliable information could be produced, the development costs for each application would be reduced significantly. Furthermore, it would provide a basis from which to partially (or wholly) support *standardized* knowledge based representations.

The introduction of reusable knowledge components is a convenient mechanism for bridging domain applications. An instrument capable of accommodating the various knowledge structures encountered in any application would require a representation that focuses on the structure of information. The content of these structures could then be filled in later by the appropriate domain experts. Surprisingly, very few knowledge acquisition tools have been designed with this type of approach and even fewer to address multiple application domains.

One knowledge acquisition system providing a domain-specific structure, KLAMShell [Cochran, 1988], was developed to aid in the construction of knowledge bases for maintenance and troubleshooting. The shell elicits information via subgoal satisfaction in a depth first manner to retain a context focus on the knowledge structure. Each goal defined is decomposed (via "push" menus) into a series of subgoals. The bottom-most nodes are then transformed (via "pop" menus) into actions, such as questions or instructions, to be used in the final system. This process continues until all goals have been specified. The system is, however, domain specific and the generality of its guidance is limited to only a *small* subset of the domain, thus restricting its use. Another system that may be classified as a general purpose knowledge acquisition shell is PROTEGE [Musen, 1989]. It is a tool capable of generating other knowledge acquisition systems using planning entities, task level actions, and input data. The methodologies used by PROTEGE separate the modeling of a domain (i.e., control structure) from the application knowledge, thereby customizing each system. However PROTEGE relies on a set of fixed templates to elicit information and this will limit the range of systems that can be produced.

From this perspective, KASH (Knowledge Acquisition SHell) has been proposed as a domain-independent knowledge acquisition shell. The shell will provide a general purpose (reusable) environment for encoding the problem-solving methods of domain experts. A set of three independent modules (Figure 1) have been defined in KASH that will operate within the top-down (analysis) and bottom-up (synthesis) constraints of various applications. These modules are: *Concept Formulation*, for eliciting knowledge from domain experts and structuring it into concept maps; *Knowledge Analysis*, to verify the concept maps and cross check any inconsistencies found; and *Rule Generation*, to produce the rule sets for the target expert system shells based on the concept maps. Temporal knowledge will be generated by the rule structures and stored in the concept maps. A time-box will supply a graphical view of the time intervals specified and a time-line analysis will show the instances of time for a particular execution of the system. The question dependency graph (QDG), which is responsible for guiding the interview sessions, will supply the control structures used in the concept formulation module. Control structures, defined *a priori* by knowledge engineers, are necessary to obtain the content knowledge from the experts. The utility of the QDG allows the control structures to be generated as needed and variations of the graph can easily address new applications.

### *Organizing Knowledge in KASH*

The basis for acquiring knowledge in KASH is by organizing the cognitive processes an expert formulates about a particular domain. The mechanisms to organize the processes must be tailored to the idiosyncratic representations created by

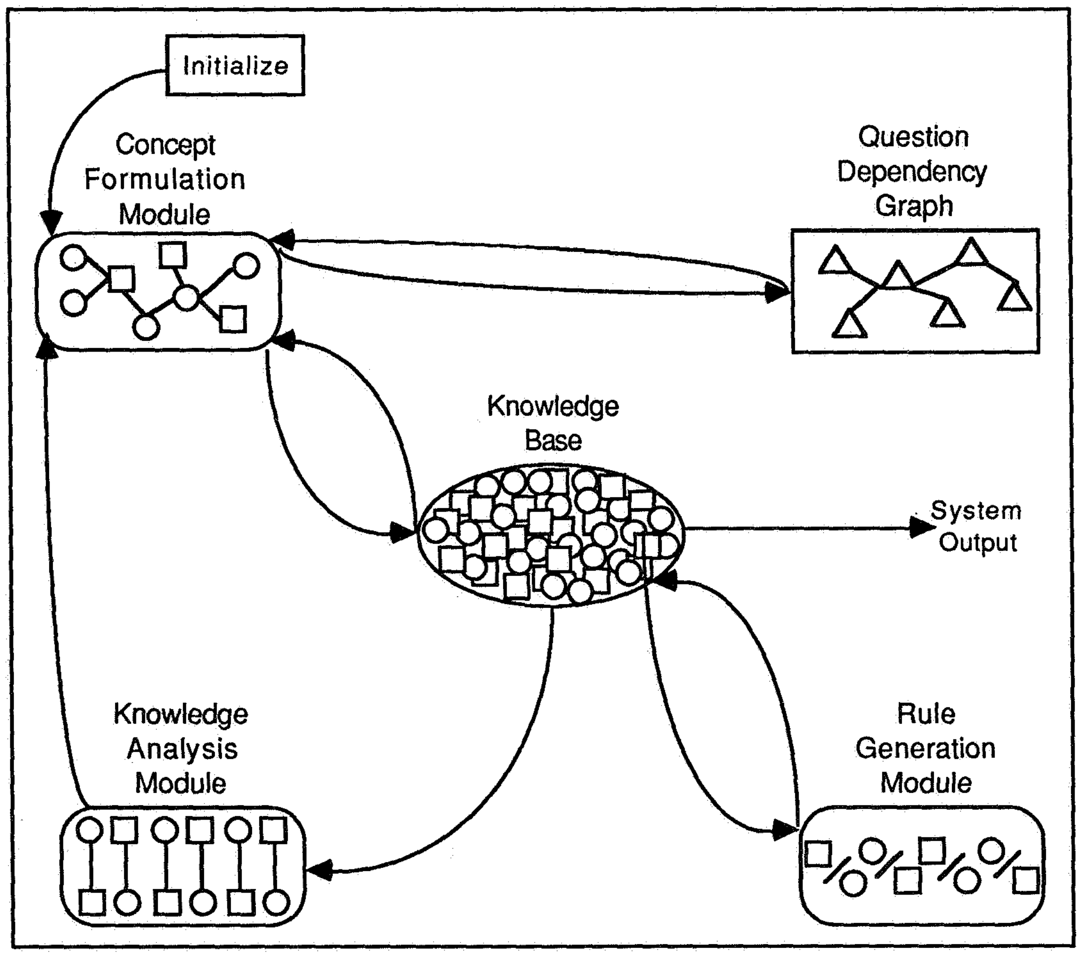

Initialize
Concept Formulation Module
Question Dependency Graph
Knowledge Base
System Output
Knowledge Analysis Module
Rule Generation Module

*Figure 1. KASH Architecture*

the experts. However, experts tend to express their knowledge in unstructured and nondeterministic formats and difficulties arise when the knowledge must be converted into a reliable and useable format for expert systems development. To properly address these representation problems, the experts must be allowed, with minimal constraints, to describe their cognitive processes with respect to the intended application. Typically, the experts will define concepts that represent personal observations in the domain to be modeled. Concepts are defined as symbols (text or image) that capture the meaning or intention of objects and events of an environment [Ausubel, Novak and Hanesian, 1978]. For example, an "electrical system" is an object, and "charge battery" is an event. In essence, a concept is a *simplified* and *generalized* description of reality for a particular level of abstraction [Novak and Gowin, 1984].

Concept formulation may be defined as the process of extracting the characteristic features, termed criterial attributes, of the objects or events. The criterial attributes are what uniquely distinguish each concept (e.g., size, color, shape). A set of common features and the degree to which the features are accepted by the experts will determine the regularity (i.e., meaning) of a concept. The regularity implies precision on the definition of the concept so that misunderstandings can be minimized, see [Novak and Gowin, 1984]. Criterial attributes also allow concepts to be grouped or linked together to form concept maps that can represent the conceptual and structural knowledge of the domain experts. A concept map is simply a hierarchical taxonomy of concepts. The hierarchy supports the natural subsumption of concepts where broad concept definitions are depicted at the top of the map, and more detailed and concise concepts at the bottom. The concept hierarchy is analogous to the familiar object hierarchy successfully applied in expert systems development. Concept maps promote the reuse of concept definitions by allowing the use of existing concepts (e.g., door) in different contexts (e.g., house, car). Such a virtual feature is natural for multiexpert integration because it provides support for a modular structure and the development of a library facility of reusable concept definitions across domains where applicable [McGraw and Westphal, 1988]. The graphical nature of concept maps also allows multiexpert conflict to be made explicit, thereby excising the knowledge structure of faulty linkages and misconceptions [Westphal and Reeker, 1990].

The concept map in KASH has been extended to include base facts [Westphal and Tran, 1990]. Base facts have their own definition because they are fully instantiated (e.g., parts, ingredients, symptoms) and do not require further justification for their existence. Base facts, therefore, provide logical grounding for reasoning performed over the concept hierarchy. Base facts are measurable and observable in the context of the concept, and support the criterial attributes of a domain. The criterial attributes are necessary for distinguishing between concepts and structuring questions into categories during concept formulation. The criterial attributes also facilitate the rule generation module because they support variable entities (e.g., size is big, color is red) that can be compared, contrasted, or combined with other criterial attributes to form the rules produced by KASH.

### *Question Dependency Graph*

The development of a concept map requires that the experts be interviewed using questions pertinent to the content and structure of the problem domain. As [Novak, 1989] stipulates, the sequencing of questions presented to the expert should be tailored or grouped into specific sets of knowledge, and the range of these sets should address questions at broad superordinate levels and become more narrow and precisely defined at the base level. In KASH, the questions permissible to ask of the domain experts during the development of the concept map will be specified by the knowledge engineers and represented in a question dependency graph (QDG). The QDG is abstractly based on the six types of questions as defined by LaFrance [1987] where each question type will decompose the QDG into a category of queries (broad and specific) to be asked with respect to conceptual definition. The six types defined are:

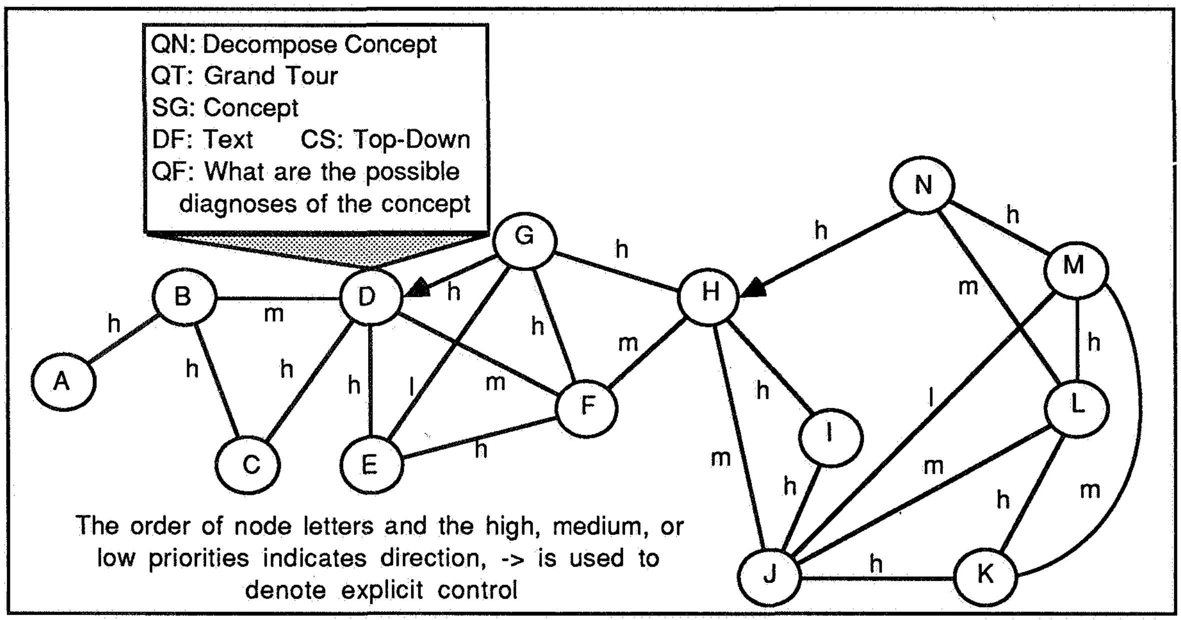
- *Grand Tour*, to identify domain and subdomain boundaries, e.g., "What is the purpose of this system?"
- *Cataloging the Categories*, to support, organize, and structure the concepts, e.g., "Can the concepts be ordered?"
- *Ascertaining the Attributes*, to specify values and ranges of the criterial attributes for concepts and base facts, e.g., "What values can attribute assume?"
- *Determining the Interconnects*, to define the causal relationships between concepts, e.g., "Does base fact support the concept?"
- *Seeking Advice*, to obtain expert recommendations and determine special conditions e.g., "Does concept contain any base facts?"
- *Cross Checking*, to compare and contrast information in the concept map, e.g., "What value of attribute is out of range?"

During the development of a QDG, a knowledge engineer will specify the questions to be asked of the domain experts. The questions will be formulated to separate the control knowledge from content knowledge of the domain (analysis or synthesis). Thus, the QDG is a form of meta-knowledge that specifies the information about how to elicit domain-level knowledge from the experts. The meta-level knowledge of a QDG implicitly tends to be high-level in nature and can be constructed as follows:

- The first version of a QDG will be obtained from a library of QDG bases. The libraries supplied to the knowledge engineers will be provided with the initial version of KASH. The QDGs will be the result of a study of the elicitation process for a particular domain (analysis or synthesis) and will cover the basic acquisition of rule constructs, temporal intervals, and a variety of generic representation mechanisms.
- The knowledge engineers will further refine and specialize the existing QDGs to meet the requirements of their application. Therefore it will be important for the knowledge engineer to understand the basic structure of the domain to effectively define questions, categorize the questions by type, and determine the control sequence of the questions. Information can be obtained through a domain analysis of technical literature, existing implementations, surveys, and system requirements. Inefficient specifications at this phase can lead to poor interview sessions with the domain expert, who will supply the content knowledge (responses to the questions).

In the QDG, the questions are expected to be represented as node structures and will be linked to other node structures to determine the sequencing. The node structure design will consist of a question frame, question type, selection guide, and a variety of support attributes. The question frame will contain the actual text generated during a query. Questions are developed by the knowledge engineers about the control structure of the domain through textbook information, job analyses, and previous case studies. The questions derived will be used to respectively refine or compose the concepts in an analysis or synthesis domain and will be continually refined through interaction among knowledge engineers and domain experts. The six question types previously defined will be used to ensure the scope of the questions are limited to the current query category. By doing so, the focus of the experts will be on the particular task level being modeled. The selection guide will be satisfied when the knowledge engineer specifies the response expected from the question frame with respect to one (or several) of the system objects specified in KASH. These objects are expected to be a prioritized taxonomy of concepts, base facts, edges, attributes, and standard representation mechanisms (i.e., integers, reals, text) that will help resolve which question to ask.

The edges will form the explicit relationships between the node structures and determine which tuples of questions may be presented to the domain experts. A tuple of two questions is interpreted to imply: if question one is asked, then question two may be asked. There will be many-to-many relationships between the node structures, thus forming a large combination of questions tuples. The edges will be coupled with edge weights to determine the ordering of question preference, a form of prefiltering. These weights will be displayed as a range of numbers or as an alternative representation.



The order of node letters and the high, medium, or low priorities indicates direction, -> is used to denote explicit control

*Figure 2. Sample Question Dependency Graph*

Figure 2 shows an example of a partial question dependency graph for a circuit fault diagnosis. Each circle represents a unique question (defined by a knowledge engineer) that may be asked during a knowledge acquisition session. The box represents the node structure (only one is detailed) for a particular query scheme. The question name (QN) is Decompose Concept, the question type (QT) is Grand Tour, the question frame (QF) is defined as "What are the possible diagnoses of the concept?" (variable terms in the question will get instantiated during execution), the selection guide (SG) is expecting the response of the domain expert to be a concept, and a miscellaneous collection of support attributes helps to specify display formats (DF) and the control strategy (CS). The links between the nodes are labeled with high, medium, and low priorities to indicate the control paths the system should pursue. In this example node D is connected to nodes E and F. The heuristics defined in the concept formulation module will select the better candidate question (E or F) to ask based on the state of the concept map.

A QDG editor will be used for the construction of the question graphs. The editor will allow the knowledge engineers to construct specialized question bases through QDG link specifications. New questions may be introduced into the graph at any time. The set of base questions will always be available, however it is the format in which they are connected that will give a system its functionality. This combination of nodes and links will allow KASH to be used across a large range of applications and will prove to be very versatile and powerful throughout the knowledge modeling process. As the QDG changes, so will the system, because the QDG is the canonical guidance referenced throughout the three modules of KASH.

As previously mentioned, the KASH system has been designed as a set of three (3) independent modules to support a reusable knowledge environment. The modules are *Concept Formulation*, to obtain and structure knowledge from an expert; *Knowledge Analysis*, to validate the knowledge structures; and *Rule Generation*, to produce a set of rules based on the acquired knowledge. Temporal knowledge is addressed explicitly by KASH through the rule generation module. Each module will be discussed in detail below. Additionally, a set of support modules will also be defined to enhance the user interface.

### *Concept Formulation*

Concept formulation, necessary to acquire the knowledge structure from a domain expert, is interleaved between a top-down and bottom-up elicitation strategy guided by the QDG. The top-down refinements are focused on acquiring the abstract concepts that are used to define the components of a domain. These components are in turn supported by bottom-up facts that represent the logical entities or extension of the concepts. The use of top-down and bottom-up strategies in concept formulation reflect the structures that exist in the different levels of knowledge and the control nature of the analysis and synthesis classes of problems. Ausubel [1978] addresses these abstractions in his propositional learning theory. The analysis problems involve identifying sets of objects based on their features. Synthesis problems construct a solution from component pieces or subproblem solutions. Figure 3 is a section of the top-down (and bottom-up) process trace of a concept formulation based on the QDG for a circuit fault diagnosis. The index (A-N) for each instantiated question text shown (i.e., variables are bound within the context of the question) corresponds to a node in the QDG shown in Figure 2.

The concept formulation module will iterate through six stages during the construction of a concept map. These stages are responsible for selecting a concept from the graph, establishing base facts for the concept, formulating a question to apply to the concept, generating the question, then presenting and accepting the results from the expert. The six stages are described below and their logic flow is displayed in Figure 4.

A)Define the terminology:
Top-down terms(s) :> diagnosis.
Bottom-up terms(s) :> symptoms.

B)What is the purpose of
this diagnosis?
:>circuit fault.

C)Define any symptoms of a circuit fault.
:>manifold threads.

D)What are the possible diagnoses of a circuit fault?
:> control circuit failure.
:> sensor fault.

E)Can these be ordered?
:> no.

F)Is manifold threads a symptom of control circuit failure, sensor fault?
:> sensor fault.

G)Are there any other symptoms of sensor fault?
:> sensor mounts.

D)What are the possible diagnoses of sensor fault?
:> broken sensor.
:> sensor grounding.

E)Can these be ordered?
:> no.

F)Is manifold threads a symptom of sensor grounding, broken sensor?
:> sensor grounding.

F)Is sensor mounts a symptom of sensor grounding, broken sensor?
:> sensor grounding.

G)Are there any other symptoms of sensor grounding?
:> voltage test.

D)What are the possible diagnoses of sensor grounding?
:> none.

D)What are the possible diagnoses of broken sensor?
:> none.

H)What are some attributes of sensor mounts?
:> connectors.

I)What value(s) can connectors of sensor mounts assume?
:> loose, broken, grounded, working.

J)Is connectors used in sensor grounding?
:> yes

K)What value(s) of connectors would make sensor grounding succeed?
:> grounded, working.

L)What value(s) of connectors would make sensor grounding fail?
:> loose, broken.

M)Is this a default or inferred value?
:> inferred.

N)Can connectors be related to other attributes?
:> no.

*Figure 3. Dialog Trace of Sample Interview Session*

Heuristic Criteria for Concept Selection - A concept will be chosen from the set of open concepts. This set represents those concepts that have not been fully explored by the QDG such that there are additional questions that may be asked about them. The concept selection will be based on several factors, including the order of importance that is explicitly specified by the user, a system defined depth-first or breadth-first selection mode, the recency of a concept definition, the level of attributes defined, the number of established links to base facts, the number of derived (children) concepts, or a concept of enumerated type.

Establish Base Facts - The first step taken after the concept has been selected will be to elicit the base facts that support the concept. The base facts define the fundamental units of the domain. New base facts will be specified by the user and the supportive links established to the selected concept. User selected base facts will be passed down to the selected concept from a parent concept. Note, parent concepts must resolve all supportive links before the concept map is considered complete.

Select Questions - The list of applicable questions for the current concept will be calculated by obtaining all the edges attached to the previous question in the QDG and placing them into a list. A duplication filter will be applied to the list to resolve any conflicts found in a list of previously asked questions maintained as a catenation of the concept name and the question identifier. The list can then be processed through some ordering filters that act on the edge weights, selection guide matching, and group division types. The product of these filters will

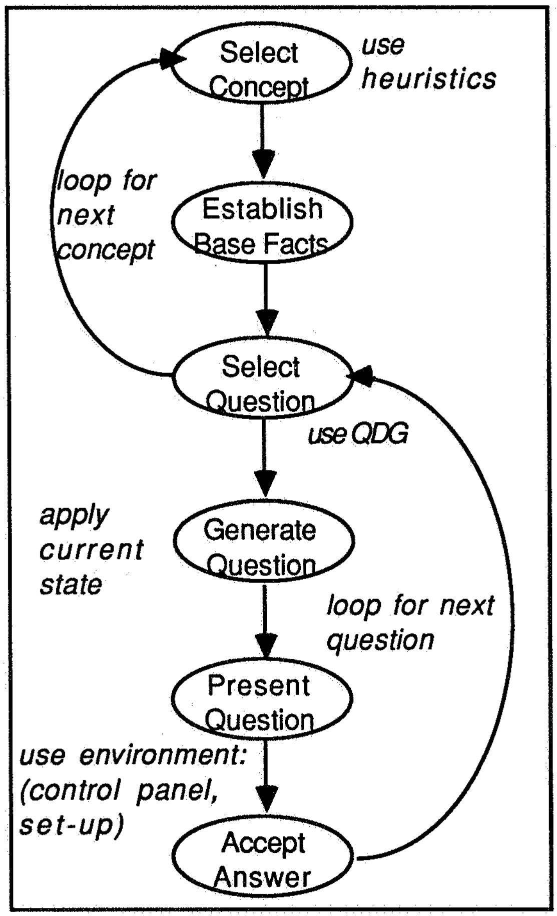


*Figure 4.*

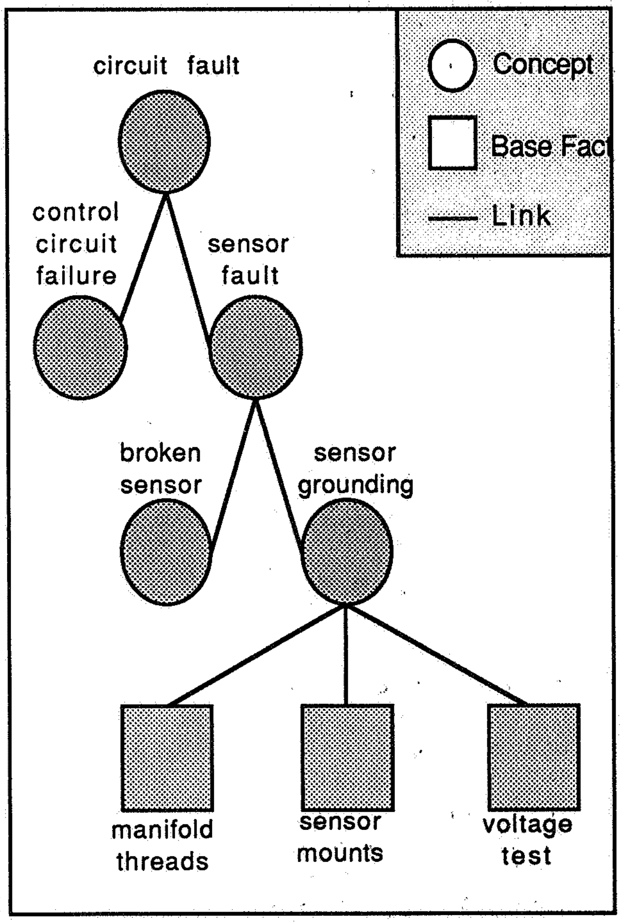


*Figure 5.*

result in a refined list of questions to be asked.

Generate Question - The product of the generate question stage will be the selection of a single query node made from the refined list. This list of applicable questions for the current concept will be acted on by a function of the question types and other system factors. A priority scale will be produced and the query with the highest rating will be selected.

Question Presentation - After a single question has been selected it will be presented to the user to solicit a response. The control panel will be consulted for the appropriate terms (top down/bottom up), display formats (bar graph, text, pie chart, menu), and concept graph information (attributes, types, links).

Answer Questions - The expert will respond to the question through either text (e.g., attributes, concepts, base facts, names, or values) or concept graph management facilities (e.g., graphics or node selection).

The output expected from the concept formulation module will be a concept map representing either a top-down, bottom-up, or combination process model. The concept map will represent the observations of domain experts as guided by the control structure specified in the QDG. Figure 5 shows the concept map for the circuit fault example. If the information acquired does not fully define the range of an expert's knowledge, the QDG may be reconfigured to accommodate the missing components. From this point the information gleaned from the expert in the form of concept modules and base facts can be verified by both the system and the domain experts. Several structure analysis techniques, described below, have been defined to accommodate this task.

### *Knowledge Analysis*

The knowledge analysis module will be a hybrid of several interactive subsystems that are used to validate and refine the concept map. During validation, the knowledge acquired from the concept formulation module will be analyzed to bring out inconsistent, incomplete, and unjustified information that may occur in particularly large systems or from the use of multiple experts. In refining the concept map, the system will look for ways to enhance the structure of the model through the application of a variety of techniques. Several proposed techniques are described below and in the accompanying Figure 6 (a-e).

Cluster Analysis - Cluster analysis will detect cases in which large numbers of concepts are derived from (or clustered around) a more general concept. It suggests that the general concept is too broad, and it should imply several new concepts where the other concepts can be derived *from*, or some derived concepts should be promoted *to* the intermediate level concepts between the general concept and others. The introduction of intermediate concepts helps structure the knowledge elicitation stage and reduces the complexity of the acquired knowledge. Figure 6a.

Entailment Analysis - The basis for entailment analysis stems from the attributes associated with the concepts. The technique will make the entailments between concepts obvious and will indicate that a concept can be subsumed by another concept because it is derived by another concept that has no other derivation. Figure 6b.

Relative Analysis - Relative analysis is based on the reuse of concept modules in the graph. A concept associated with a base fact will be selected for review. The analysis technique will propagate up the structure to a level outside the immediate hierarchy associated with the concept, and query the expert, via the QDG, if it can be used in the adjacent paths. This type of analysis will support the use of virtual structures and serves as a form of memory cue entailment for additional structure refinements. Figure 6c.

Subcomponent Analysis - A base fact will be selected from an active path in the concept structure. The path will consist of the base fact and all the concepts that comprise the path to the root node. The base fact will be compared to each of the concepts that subscribe to the path and questioned by the QDG for relevancy against the concept. Negative responses will indicate inconsistencies in the structure and will be resolved through further QDG examinations. Figure 6d.

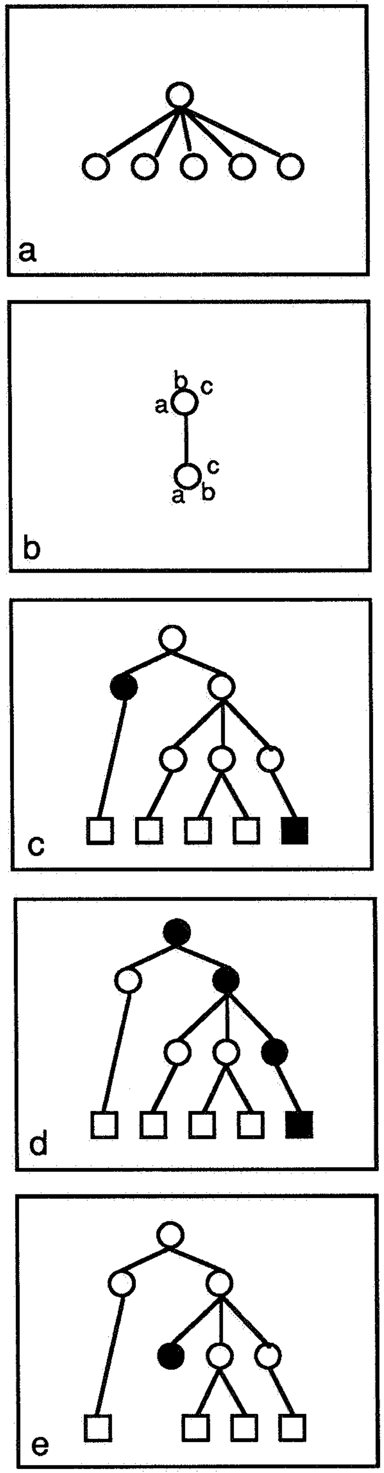


Figure 6 (a-e).

Conceptual Analysis - This attempts to enforce the principle that every base fact must be attached to a concept that is not further decomposed by additional concepts. If any such case is encountered it can be assumed that there will be additional conceptual levels that have not been defined. Additionally, if a concept has been decomposed into another single concept, then the linear formation will invoke the analysis to collapse both into a single concept structure. Figure 6e.

Knowledge analysis will be interactive or selectable. In the first mode, *interactive*, a number of knowledge analysis techniques will be selected from a menu before the concept formulation module is executed. As knowledge is acquired the techniques will be automatically invoked to yield analysis results. The results will be shown in a side window with the appropriate marks to indicate the different degrees of importance. In the second mode, *selection*, the knowledge analysis stage will be entered after a knowledge elicitation session. The manner in which the analysis results are shown will be the same as in the previous mode, but more thorough analyses can be applied.

### *Rule Generation*

The third module of KASH will be used to support the final process of knowledge acquisition, rule generation. This module will create a set of rules that can be integrated into a third party expert system shell. The three major factors that will influence the generation of the rules are the concept map, rule generation strategy, and target expert system shell.

First, the concept map will be extended to include any terms (criterial attributes) necessary for the production of the rules. New attributes will be created and manipulated by *attribute<->attribute*, *attribute<->concept*, and *attribute<->value* questioning schemes defined in the QDG. The information provided by the experts in response to the questions will specify the attributes of a concept, the relationships of the attributes to other attributes, and the range of values the attributes may assume. Attribute definitions will be initiated for each base fact in the concept map. The definitions will be propagated to the concepts via an inverted inheritance scheme with question filters to terminate any attribute stream. The attribute relationships in the concepts will represent rules. The rules will be local to the concept for which they are defined and may be classified as either *default*, where new logical relationships between attributes will be specified (e.g., set the temperature of material to equal the temperature of the gas), or *inferred* where the system must calculate the value of the used attribute (e.g., does the temperature of the material equal the temperature of the gas). The inferred values will be present in the rule premises and the default values will typically appear in the consequents.

Second, in the rule generation strategy, the local rules will be used by a series of high level top-down and bottom-up strategies that determine the direction in which final rules are to be developed. The top-down strategy employed in generating rules will be based on a concept and the links to its supportive concepts. The bottom-up strategy will direct the rule generation based on the lower level concepts to the higher level concepts that they support. This design will provide both forward and backward chaining rule specifications for integration into the expert system.

Third, the characteristic or model of the target expert system shell language will influence the different ways in which an optimal rule set may be generated. This will depend on the different representation schemes, control strategies, daemons, triggers, etc. encountered in the different expert system shells. For example, instead of explicitly stating separate rules as would be necessary in a pure production system such as OPS-5, the number of rules generated can be reduced in an object oriented system such as NEXPERT, CLIPS, or KEE due to the abstraction of frame structures.

```
Rule 1                                            Rule 2
IF level_of_voltage_test < 3                      IF level_of_voltage_test >= 3
THEN position_of_sensor_grounding = "open"AND level_of_voltage_test < 5
                                                  THEN position_of_sensor_grounding = "closed"

Rule 3                                            Rule 4
IF connectors_of_sensor_mounts = "loose"          IF connectors_of_sensor_mounts = "broken"
THEN cond_of_sensor_grounding = "poor"            THEN cond_of_sensor_grounding = "poor"
AND recd_of_sensor_grounding = "tighten"          AND recd_of_sensor_grounding = "replace"

Rule 5                                            Rule 6
IF connectors_of_manifold_threads = "dirty"IF connectors_of_manifold_threads = "corroded
THEN cond_of_sensor_grounding = "poor"            THEN cond_of_sensor_grounding = "poor"
AND recd_of_sensor_grounding = "clean"            AND recd_of_sensor_grounding = "replace"

Rule 7                                            Rule 8
IF cond_of_sensor_grounding = "poor"              IF cond_of_sensor_grounding = "poor"
AND position_of_sensor_grounding = "open"         AND position_of_sensor_grounding = "closed"
THEN status_of_sensor_fault = "true"              THEN status_of_sensor_fault = "true"
AND action_of_sensor_fault = "shutdown"           AND action_of_sensor_fault = "maintenance"
AND recd_of_sensor_grounding =                    AND recd_of_sensor_grounding =
       recd_of_sensor_fault                              recd_of_sensor_fault
```

*Figure 7. Partial Rule Constructs Generated by KASH*

Figure 7 shows the partial set of rules that have been created for the circuit fault diagnosis example. Before these rules are converted into a target expert system format they must be reviewed by a series of analysis techniques to ensure there are no redundant, conflicting, or circular rules.

The outcome of the rule generation system module will be a knowledge base of objects, relationships, and rules that could support a variety of expert system applications.

### *Representation*

The underlying information detailed during the construction of the concept map, QDG, and rules are expected to be stored in a frame-structure representation. The frame-structure, originally proposed in the 1970s, supports the modularity necessary to easily add, modify, or delete information from the knowledge structure. It is the degree of modularity encountered in a system that directly promotes the decomposability of an application [Simon, 1969], as may be seen during the construction of the concept map. The KASH architecture has been designed with minimal dependencies between its representations such that all question nodes in the QDG and sections of the concept map can be extracted, interpreted, and reused in other applications. Furthermore, the slot values associated with frame-structures will provide KASH with one mechanism to confirm a complete specification of the rule constructs.

### *Temporal Knowledge*

Representing and reasoning over time is an important factor to consider when developing expert systems, especially in domains such as evaluation, planning, and scheduling. Although there has been a large effort of work associated with time, very little research has been conducted with respect to temporal knowledge acquisition. The KASH system will address this unfledged technology by eliciting *numerical* and *symbolic* time references through the use of a time-box facility and time-line analysis. It is expected that temporal knowledge will be elicited (when required) through an extension of the rule-generation module.

The hierarchical structure of concept maps forms a type of temporal circumscription where absolute time intervals (*numerical*) can be applied without ambiguity in meaning. The specification of these intervals is straightforward and done by the domain expert for all relevant concepts encountered in the graph by limiting the duration specified for a concept to its lineal descendants. When a time interval has been stated for a primary concept, any subsequent intervals stated for its dependent concepts must not exceed the initial interval; the duration of the parts must be equal to or less than the whole. For example, if concept A is supported by concepts B and C and the duration set for A is <u>five</u> minutes, then the total time to execute B and C should not exceed this interval. Start, end, and latency times are required for numerical intervals. The decomposable nature of concept maps supports temporal relations (*symbolic*). Concepts are composed *of* or derived *from* other low or high level descriptions (concepts or base facts) of the application. Thus, when describing the temporal relationship between two concepts, Allen's [1983] enumerated set of time primitives may be used. For example, concept B must occur <u>during</u> concept A. Utilization of these primitives also provides a set-theoretic method of reasoning about temporal relationships. Both the absolute and relative time constructs will be easily accommodated by the slots associated with the frame representation used by KASH.

The questions concerning the representation of time constructs will be defined in the QDG jointly under the determining-the-interconnections and ascertaining-the-attributes classifications. Examples of such questions will include "Can the concepts be ordered?" or, "Does concept_x occur before concept_y?" or, "What is the start time of concept_z?" A time-box has been defined (Figure 8) that will depict the results of the temporal specifications for the concepts in a graphical manner. The horizontal axis of the time-box represents the forward notion of time and the vertical axis is used to display the depth or level of the concepts. Recurring patterns will be represented in the time-box as open-ended (broken) time slices and are defined as those instances where the activities occurring during a particular series of time units are repeated more than once (see intervals E, L, & O). In KASH the repeating section will be related to a triggering event (i.e., a specific attribute value in a rule sequence) in the superordinate time interval. For every instance the event is encountered, the corresponding activities will repeat for their defined intervals. All subsequent time executions will be adjusted for multiple successions of a repeating event, thus maintaining a dynamic consistency across the temporal span. The period of 'Mondays' in a particular month is an example of a recurring pattern as defined by [Ladkin, 1988] and would be composed as the union of each unique 'Monday' for that month interval. The triggering event here may be the requirement of a staff meeting every Monday morning.

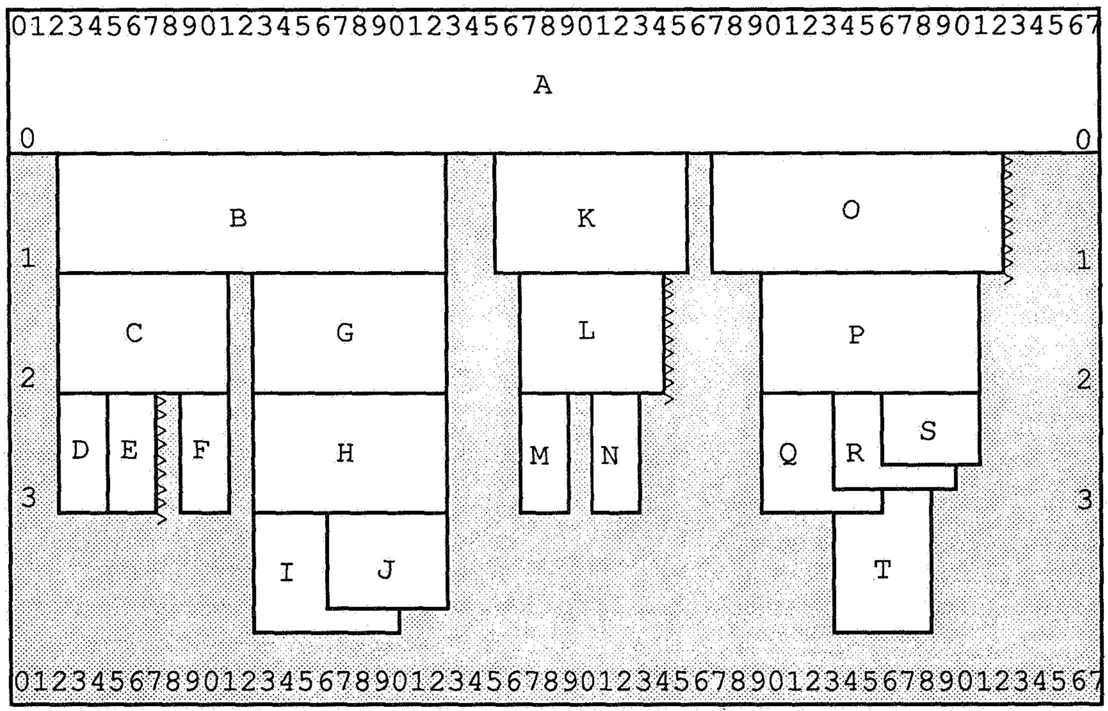


*Figure 8. Time Box*

A time-line analysis complements the time box. Whereas the time-box will be used to define time for the structure of the application, the time-line will show the actual instances (persistence) of time for an execution of the system, see Figure 9. A unit of time will represent an interval where the state of the system will be allowed to change. Many instantaneous changes can occur within a time unit, but it is the final values that will be recorded and stored in the time-line analysis. For example the rule "If the temperature_of_water > 212 for more_than 3_minutes Then set status_of_water = boil" would examine the current state of the time-line for the slot *temperature* of the concept *water* and determine if the value has been greater than *212* (unit=degree) for a duration of time exceeding three minutes. If the time-line analysis supported the antecedent condition, the slot *status* of *water* would be set to equal the value of *boil*. The time-line analysis in KASH will prove similar in functionality to that of the temporal network in ONCOCIN [Kahn, 1985] and the time mapper in KAT [Geesey, 1988]. As is expected with all results generated in KASH, the temporal information will have a series of analysis techniques that are expected to detect conflicts and unreachable situations caused by misrepresented time intervals.

### *Supportive Modules*

There exist several external features that will be added to the basic functionality of KASH. These features are intended to extend support to the knowledge acquisition process and supplement the work of the QDG. The following represent a subset of the planned features:

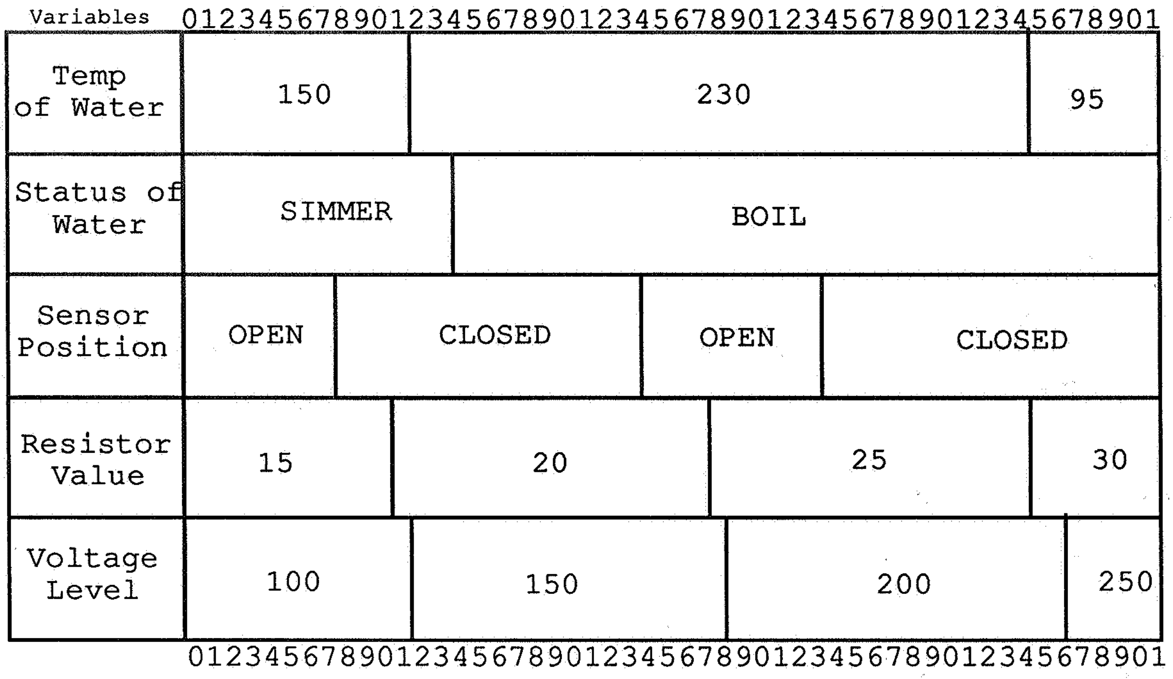


*Figure 9. Time Line Analysis*

- An enhanced control panel will contain a multitude of user-defined parameters to coordinate the elicitation sessions. The panel will contain the terms used for the top-down and bottom-up definitions. Pre-logical settings for which top-down entries are best suited for the bottom-up selections will be available. The search strategies (i.e., depth-first, breadth-first) can be set from the panel. A question list will be defined in the panel and used to disable groups of questions for the session (usually low priority settings). Additionally, the analysis techniques will be made interactive or selectable from the panel settings.

- A history window will supply a meta-command interpretation of the events occurring in the system. This will provide documentational support to analyze the elicitation session as it progresses over time. The log will be used to archive the state of the concept model for easy reconstruction at a future date. Additionally the history list will be constructed for path resolution of the virtual system hierarchy.

- A library window will provide interactive graphic access to previously defined concept hierarchies and QDGs. This facility will enable partial or complete merging of libraries and the current environment, thereby supporting the reuse of KASH structures.

- A glossary of terms defined by the expert during the elicitation of concepts and base facts will be supported as a scrolling series of menus. This facility can support the comments associated with multiple-expert development. The glossary support system can be expected to emulate a hyper-expert notation scheme where the terms defined can be a mixture of text and graphics.

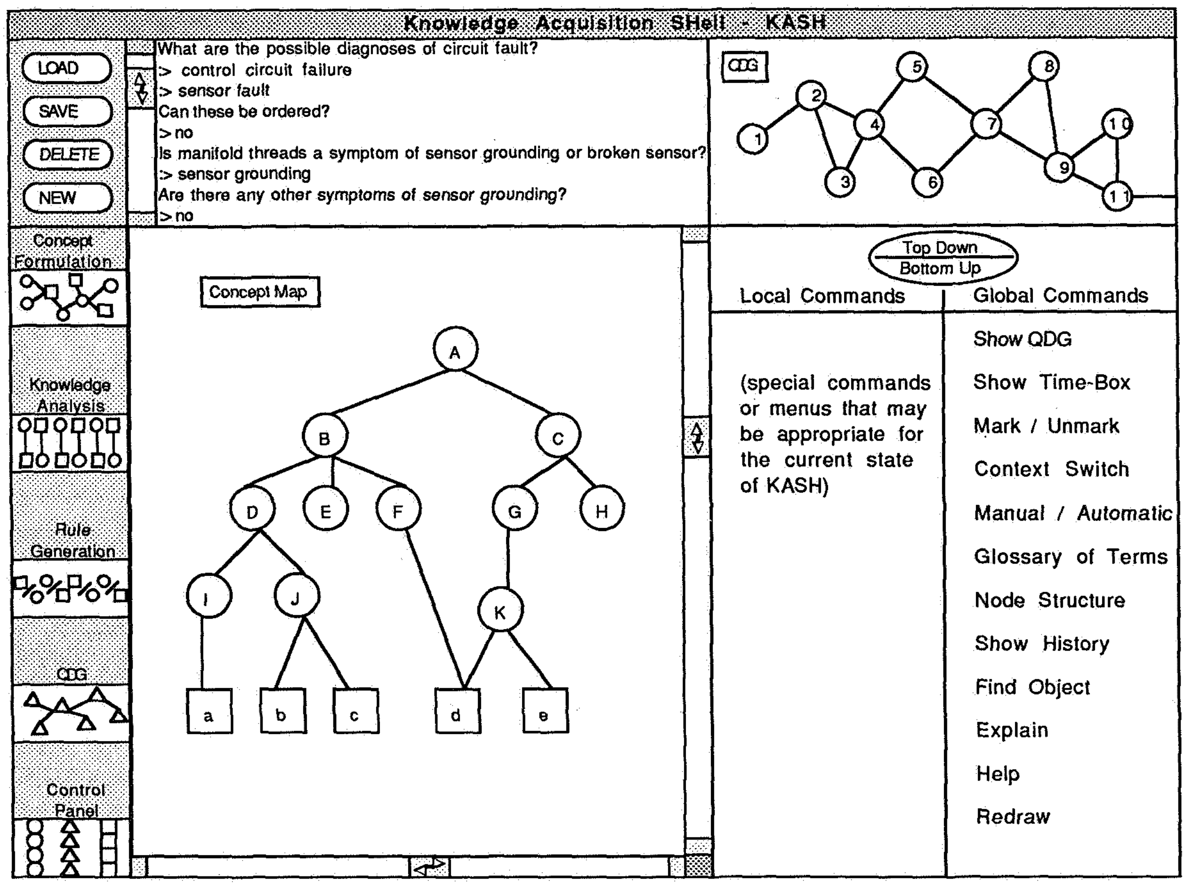
### *Implementation*

The KASH system has a considerable level of potential for expediting the knowledge acquisition process and would be most beneficial if it were made widely accessible to the knowledge engineers. This goal requires that KASH be implemented on workstations or personal computers (PCs) because of their availability to the individual users without being subjected to the limited resources of larger systems. • A graphical interface will be necessary to provide a robust environment that will be both useful and meaningful to the end users. The current state-of-the-art offers one such public domain technology, X-Windows. X-Windows is a portable, network transparent windowing system that is widely available on UNIX machines. An X-workstation or terminal (e.g., PC, Sun, MIPS, or Dec) would provide the industry standard look-and-feel interface, thus making KASH highly portable because the interface codes would not be required to be redeveloped between system architectures (see Figure 10). • The KASH system will be coded in the programming language C to achieve the primary goal of portability.

The initial target expert system shell for which KASH will generate rules will be CLIPS (C Language Production System). CLIPS was developed by the Mission Planning Group at NASA/JSC and is a very good candidate expert system shell. CLIPS provides high portability, low costs, and easy integration with existing conventional software systems. The availability of the CLIPS source code will prove beneficial because KASH will be able to treat the system *not* as a black box, but as an integrated component of the whole development environment. Furthermore, CLIPS as a C language callable library of functions, will provide a good testing platform for knowledge acquisition for embedded intelligent systems.

## Conclusion

KASH is a general purpose knowledge acquisition shell that will acquire information about a domain from an expert. Since KASH has been designed to support both analysis and synthesis applications, it may be applied to a broad range of systems including planning, design, classification, scheduling, decision support, and diagnosis where complex knowledge is often acquired from multiple domain experts. The three modules defined in KASH will provide the capability to acquire knowledge in context (i.e., customized for each domain): *Concept Formulation* will structure and elicit the knowledge from a



What are the possible diagnoses of circuit fault?
:> control circuit failure
:> sensor fault
Can these be ordered?
>no
Is manifold threads a symptom of sensor grounding or broken sensor?
:> sensor grounding
Are there any other symptoms of sensor grounding?
>no

(special commands or menus that may be appropriate for the current state of KASH)

*Figure 10. KASH User Screen*

domain expert into a concept map; *Knowledge Analysis* will validate the concept map; and *Rule Generation* will produce a rule set for an expert system based on the concept map. Additionally, temporal knowledge acquisition will be accomplished through modifications to the concept maps using a time-box and time-line analysis to display the interval units. All the information elicited from a domain expert will be guided by a question dependency graph (QDG). The QDG will be developed by a knowledge engineer to separate the control knowledge from the application knowledge. Thus, the QDG will be reconfigured and customized across applications and domain experts. Furthermore, KASH will have the resources to produce a knowledge base that can be used to generate alternative rule sets for different expert system shells.

## *References*

Allen, J.F. (1983). "Maintaining knowledge about temporal intervals," *Communications of the ACM*, Vol 26, No. 11. pp. 832-843.

Ausubel, D.P., J.D. Novak, and H. Hanesian (1978). *Educating Psychology: A Cognitive View*, 2nd edition, New York: Holt Rinehart, and Winston. Reprinted, 1986 New York: Warbel and Peck.

Cochran, E.L. (1988). "KLAMShell: A domain-specific knowledge acquisition shell," Technical Report CSDD-889-I6301-1, Honeywell, Golden Valley, MN.

Geesey, R.A. (1988). "A principled approach to tooling for temporal knowledge acquisition," BDM Technical Report, BDM/ROS-RG-06254-88. May 12, 1988.

Kahn, M.G., J.C. Ferguson, E.H. Shortliffe, and L.M. Fagan (1985). "Representation and Use of Temporal Information in ONCOCIN," *Proceedings Ninth Annual Symposium on Computer Applications in Medical Care*, IEEE Computer Society, pp. 172-176.

Ladkin, P. (1986). "Primitive and Units for Time Specification," *Proceedings of the Fifth National Conference on Artificial Intelligence*, Philadelphia, pp. 354-359.

LaFrance, M. (1987). "The knowledge acquisition grid: A method for training knowledge engineers," *Int. Journal of Man Machine Studies*, 26, 245-255.

Marcus, S. (1988). "SALT: A knowledge-acquisition tool for propose-and-revise systems," In Marcus, S. (ed.), *Automating Knowledge Acquisition for Expert Systems*. Kluwer Academic Publishers, Boston, pp. 81-122.

McGraw, K.L. and C.R. Westphal (1988). "Accommodating Multiple Expert Input During Knowledge Acquisition: Integrating Hypertext with a Knowledge Acquisition Tool," *Sixth Annual Intelligence Community AI Symposium*, Langley, VA.

Musen, M.A. (1989). "Knowledge acquisition at the metalevel: Creation of custom-tailored knowledge acquisition tools," *SIGART Newsletter*, No. 108, April. pp. 45-55.

Novak, J.D. (1989). "Helping students learn how to learn: A view from a teacher-researcher," *Proceedings of the Third Congress on Research and Teaching of Science and Mathematics*, Santiago de Compostela, Spain.

Novak, J.D. and B.D Gowin (1984). *Learning How to Learn*, New York: Cambridge University Press.

Rodi, L.L., J.A. Pierce, and R.E. Dalton (1989). "Putting the expert in charge: Graphical knowledge acquisition for fault diagnosis and repair," *SIGART Newsletter*, No. 108, April. pp. 56-62.

Simon H.A. (1969). *Sciences of the Artificial*, Cambridge, MA. MIT Press.

Westphal, C.R. and L.H. Reeker (1990). "Reasoning and representation mechanisms for multiple-expert knowledge acquisition," in *Knowledge Acquisition: Current Practices and Trends*, McGraw, K.L. and C.R. Westphal (eds.), Ellis Horwood, Chichester, England.

Westphal, C.R. and D. Tran (1990). "KASH: A general purpose knowledge acquisition shell," *Proceedings of the Florida Artificial Intelligence Symposium*, Cocoa Beach, FL.

# CAPTURING FLIGHT SYSTEM TEST ENGINEERING EXPERTISE: LESSONS LEARNED

Irene Wong Woerner
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California 91109

## ABSTRACT

Within a few years, JPL will be challenged by the most active mission set in its history. Concurrently, flight systems are increasingly more complex. Presently, the knowledge to conduct integration and test of spacecraft and large instruments is held by a few key people, each with many years of experience. This expertise is unique to JPL and not readily available through academia or industry. JPL is in danger of losing a significant amount of this critical expertise, through retirements, during a period when demand for this expertise is rapidly increasing.

The most critical issue at hand was to collect and retain this expertise and develop tools that would help ensure JPL's ability to successfully perform integration and test of future spacecraft and large instruments.

The proposed solution was to capture and codify a subset of existing knowledge, and to utilize this captured expertise in knowledge-based systems. Such systems would be available simultaneously to numerous tasks and would also facilitate knowledge transfer to a new generation of engineers.

Consequences of not implementing a solution immediately were loss of expertise, thus potentially jeopardizing JPL's current preeminence in integration and test of spacecraft and large instruments. Further consequences of inaction were either a substantial increase in cost to adequately test future missions or inadequate test programs that significantly increase mission risk.

This paper describes first year results and activities planned for the second year of this on-going effort . Topics discussed include lessons learned in knowledge acquisition and elicitation techniques, life-cycle paradigms, and rapid prototyping of a knowledge-based advisor (Spacecraft Test Assistant) and a hypermedia browser (Test Engineering Browser). The prototype Spacecraft Test Assistant supports a subset of integration and test activities for flight systems. Browser is a hypermedia tool that allows users easy perusal of spacecraft test topics. This paper will also describe a knowledge acquisition tool called ConceptFinder which was developed to search through large volumes of data for related concepts and will be modified to also semi-automate the process of creating hypertext links.

## Introduction

This paper describes an initial phase of an effort to capture a set of processes required for integration and test of flight systems, and to develop knowledge-based tools which utilize this captured expertise. This effort is part of the solution to help JPL preserve its expert knowledge of flight system test, make this knowledge readily available to less experienced engineers, and develop knowledge-based automated tools for future, more complex test programs.

## Objectives

Major technical objectives for the first year were to 1) prove feasibility and effectiveness of combining multiple technologies, including expert system and hypertext, with test knowledge and models on a large scale; 2) establish an approach and architecture to facilitate systematic achievement of goals through building the necessary generic structure, tools, processes and technologies; and 3) capture, codify and automate a subset of current processes of integration and test.

These first year objectives were met by capturing a subset of information on flight system environmental testing, and using this captured expertise in two knowledge-based prototypes which assist in test engineering functions.

## Background

Over the past 20 years, JPL's planetary program has successfully developed and tested a set of increasingly complex spacecraft. Until now, missions have been more or less serial in nature, such that integration and test of these spacecraft have been planned and accomplished by essentially the same set of key engineers and managers. Looking toward the future, two factors are converging that cause concern. First, as emphasis shifts from building spacecraft to building large-scale instruments, simultaneous test activities will be required. The number of experienced spacecraft and large instrument "test experts" is limited. Accordingly, major instruments would have test programs managed and conducted by less experienced engineers. Second, with only one exception, key lead individuals who comprise the experienced test team are all over 50 years of age.

JPL's risk of losing this capability has become more real. Last year, only 5 persons at JPL had knowledge and actual experience in managing a spacecraft test program. This year, there are only 4. There are 8 additional "test experts" of spacecraft and/or large instruments who could manage a large test program. Eleven of 12 experts identified are over the age of 50. The following chart illustrates the vast experience base, 420 cumulative years, JPL is in danger of losing.

|  | Experienced Managers |  | Test Experts (8) |
|---|---|---|---|
|  | 1989 (5) | 1990 (4) |  |
| Average Age | 58 | 56.8 | 56 |
| Total Years Experience | 168 | 132 | 260 |
| Average Years Experience | 33.6 | 33 | 32.5 |

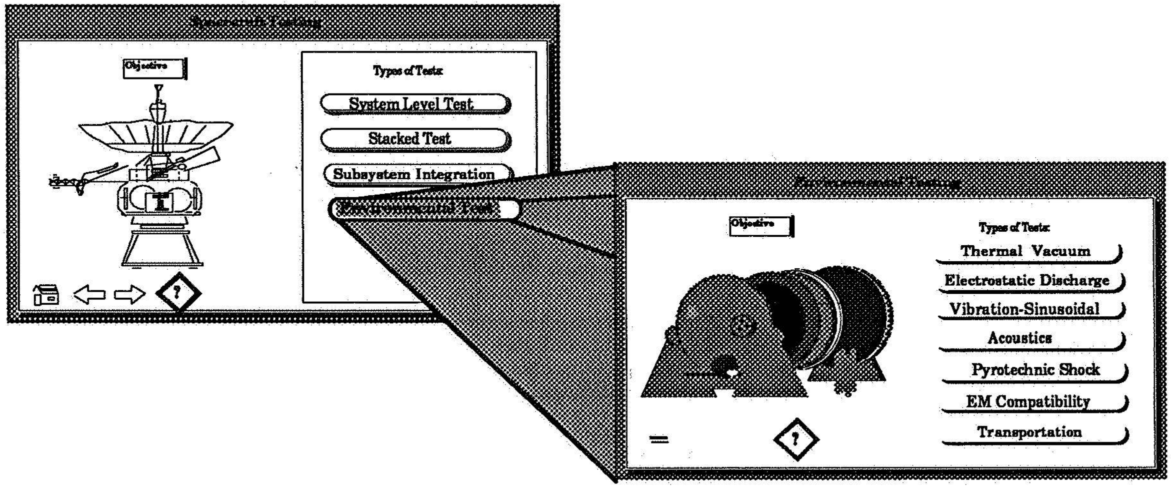
It is estimated that within 5 years, 80 percent of this expertise may retire. Within the first year, this body of experienced test managers has decreased by 20 percent, because one of our most experienced experts retired. The threat of losing our expertise is already being realized.

## Technical Background

In expert systems, knowledge and control are separated. Knowledge is concentrated into a knowledge base, while control resides in an inference engine. A knowledge-based system capturing a subset of existing knowledge on flight system testing was viable for a number of reasons. For one, expertise exists and was available. This problem, loss of I&T expertise, was faced by an organization, rather than an individual. Also, there was a sufficient amount of information needed in decision making to justify the use of a "smart" information system, and the task of application was non-trivial, yet well-bounded and easily expandable. (Vassilio83)

Hypertext is a software methodology for presenting information in a non-linear fashion. Ted Nelson, a pioneer of hypertext, defined it as "a combination of natural language text with the computer's capacity for interactive branching, or dynamic display... of a nonlinear text... which cannot be printed conveniently on a conventional page." (Nelson67) An outstanding feature of hypertext is a physical realization of conceptual links which conventional text can only symbolize. (Monk88) A textual cross-reference to a related subject is an example of a conceptual link in conventional text.

Hypertext technology allows for retrieval of precise information needed for a specific task in an easy and cost-effective manner. For example in prototype Browser, if a user wants more information on environmental testing, pointing to that topic with a mouse and "clicking" brings up this information. From there, other environmental test related information can be accessed. In fact, a user can traverse to a detailed test procedure if that is the product desired. This example is shown pictorially below.

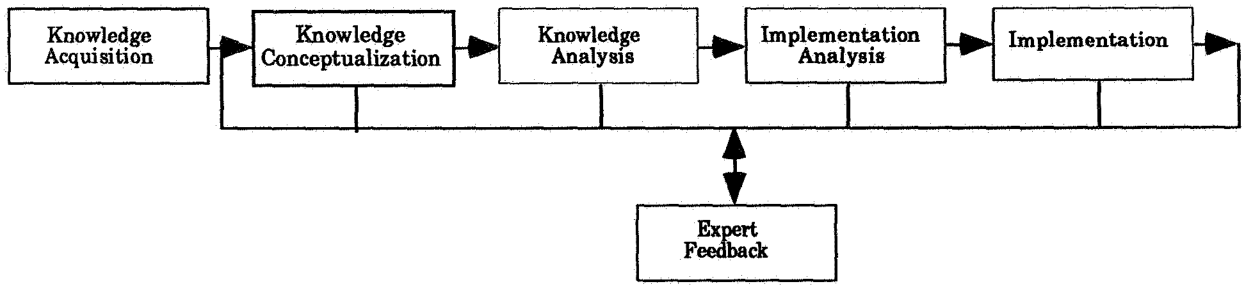


Adding hypertext technology to a rule-based system also provides an additional knowledge representation technique. The number of traditional rules required is reduced; specifically rules related to explanation. Feasibility of combining these two technologies has already been demonstrated in another JPL effort "Application of Expert System and Hypertext Technologies to Technical Document Preparation." (Wong90)

## Approach

A knowledge engineering process was developed by tailoring and combining a process described by Hayward (Hayward88). and Barry Boehm's spiral model (Stachowitz87). Hayward defined a process that categorized knowledge engineering into five phases: knowledge acquisition, knowledge conceptualization, knowledge analysis, implementation analysis, and implementation. These phases were applied to Boehm's spiral model which essentially calls for a continuous iteration of these phases around an axis of time. This spiral process is typical of rapid prototyping development which involves multiple iterations. However, a departure from traditional rapid prototyping techniques is addition of expert feedback at each phase rather than solely during evaluation of a prototype. Rather than waiting until a prototype was implemented, experts were shown meta-products for verification of correctness. Rapid prototyping is a desirable approach, because it facilitates verification of implemented knowledge for correctness and completeness by a panel of experts, developers and management. With a rapid prototyping approach, meta-products and the prototype are incrementally evaluated and their effectiveness determined. Products are then improved iteratively, based on feedback, until experts and users are satisfied with results.

Involvement of experts in every phase of development was helpful on two levels. First, we could verify the correctness of what the knowledge engineer interpreted much sooner and more efficiently than waiting until a rough prototype was implemented. Although this required more "up front" time, errors are not propagated and are easier to correct; especially when the system is still a paper product. Second, experts are involved at every step of development and feel more ownership, because experts could see progression of system development and incorporation of information and data that they supplied. The following figure illustrates the modified approach:



In the first year, a subset of existing knowledge on integration and test of flight systems was captured in a structure-free, textual information base by transcribing one-on-one interviews with domain experts. Pre-conceived structure was purposely avoided so that information gathered would not be inhibited by how information is asked or stored. This information base is comprised of Wordperfect and ASCII files. Another benefit of a structure-free format is traceability of rules to pieces of raw data these rules were derived from. Furthermore, this process ensures gathered knowledge is not lost in translation from text to knowledge structure. It is also highly desirable to perform knowledge acquisition in such a way that results are reusable beyond present goals. For example, a different knowledge structure could be implemented at a later time, and the raw data would be available rather than just rules. Since a common word processing package was used for the transcription process, no training was required. This process has resulted in a raw information or "data" base that is equivalent to about 600 pages of text. Existing documentation which also contains a lot of embedded knowledge contributed another 500 pages of text.

There were several techniques which help in knowledge acquisition. Since the information base is large, tools were needed for information retrieval. Retrieval tools are a valuable aid to knowledge engineers, because the raw information base that needs to be searched is very large. Furthermore, existing technical documentation compounds the amount of data a knowledge engineer must sort through. A tool to help search for related "chunks" of concepts for encapsulation in a knowledge representation structure enables a knowledge engineer to transform data into information more efficiently. Nevertheless, retaining a complete raw information base is important.

Commercially available information retrieval tools were used to facilitate information retrieval from such information bases. However, a PC-based prototype, ConceptFinder, was developed and also used in conjunction. Reasons for developing ConceptFinder include an ability to search through either WordPerfect or ASCII files from within any application. ConceptFinder is a terminate and stay resident (TSR) program and related concepts are defined in a thesaurus-based fashion. For example, related concepts of thermal vacuum testing are hot/cold nominal testing, tailoring of chambers, and bake-out tests. Searching on thermal vacuum testing will also identify portions in the raw information base that mention bake-out tests or tailoring of chambers.

Furthermore, ConceptFinder was easily extendable to include hypertext authoring capabilities based on the same related concepts thesaurus. ConceptFinder gives a list of related concepts already defined in hypertext and facilitates establishment of links by generating the code needed to link a topic. This authoring capability paves the way for automatic generation of hypertext links later. A basic research question that still exists is how to establish context for automatic hypertext link generation. A technique we will try, within a limited domain, is statistical analysis of related words clustering. For example, if the word "power" is surrounded by words like "ground" or "voltage," links chosen would be different than "power" surrounded by words like "politics" or "money." However, this is not an effective method for establishing context in a general purpose or knowledge environment. (Salton89) Consequently, this is at best an inefficient work around until a better method is available.

After conducting several high level interviews on flight system test engineering to determine the general scope and relationships between large conceptual tasks, overall knowledge on integration and test was decomposed into logical modular units such as "integration of subsystems" and "environmental testing." Breakdown of a large task into manageable units help ensure objectives and design are robust, and easier to expand progressively and systematically in scope. The next level is to focus on one specific area; keeping in mind where this piece fits in the "big picture." This process helps codify what and how we do things. Stages of interview progress from orientation to identification to analysis. These stages represent the levels of detail that knowledge acquisition requires. Orientation is a grand tour which establishes an overall structure. Identification focuses in on one area, and analysis provides the rules and underlying relationships that exist for this one area of knowledge.

During the first year, the subdomain of focus was environmental testing, specifically thermal vacuum testing. This is a defined task which represents a subset of critical information necessary in most flight system test programs. Initial knowledge acquisition also included establishing a common language between expert and knowledge engineer. This common language was necessary to gradually minimize the ambiguities that accompany different and non-shared experiences. Consequently, the number of unfamiliar words or concepts diminish over time, and less time is spent in defining terms.

First year findings indicate knowledge acquisition can be categorized into three basic types: 1) past events - "I typically found dead shorts are caused by rework and miswiring;" 2) current activity - "Why do you check all grounds first? Because once the spacecraft subsystems are stacked, it's hard to access and test;" and 3) future events - "How would you test something like CRAF (Comet Rendezvous Asteroid Flyby)? I would first figure out what its mission is suppose to be, determine the kinds of instruments it has on board, etc." We found it easier to start with a retrospective question, "What did you do on x?" and integrating additional types subsequently. There are multiple reasons for this approach: 1) experts had just completed a major spacecraft project; 2) experts were not currently testing a systems and therefore, could not be shadowed to codify what they were doing and 3) future spacecraft were not adequately defined to determine details of a comprehensive test program. Consequently, the best types of questions to start with is very dependent on which activity is most immediate -- past, present or future.

To conceptualize knowledge, knowledge presentation diagrams were created. These diagrams are graphical representations of an expert's thought process as translated by a knowledge engineer. A knowledge presentation diagram is tangible proof to everyone that logically documenting expertise is possible. (Kearne90) It served as a focal point for further discussion with an expert and as a guideline for tool development. These diagrams were instrumental in translating existing data into rules, and were also used in creating hypertext links. They provided much of the control flow logic and also aided in creation of a top-level hypermedia navigation map. Studies have shown that a graphical representation of a "document's" structure vastly aided performance of a user's awareness of where they were in a hypertext document. (Simps89) This capability lessens the probability of a user getting lost in hyperspace. This condition occurs when a user has lost their point of initial reference, because of traversal through too many links.

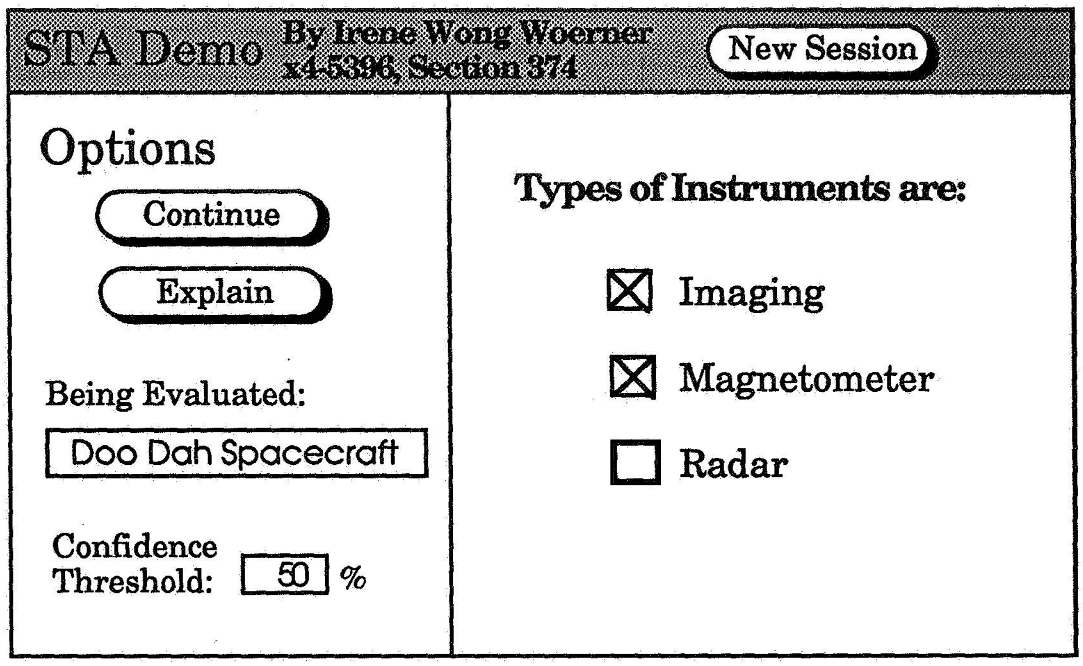
Knowledge analysis incorporated both epistemological and logical analysis. Epistemological analysis concentrates on examining structural properties of conceptual knowledge especially with reference to limits and validity (e.g. "Environmental testing includes vibration and acoustic testing, and you would use acoustic testing to simulate a launch profile."). Logical analysis determines control structure, and what factors are responsible for inference making (e.g. "If you have an imaging device, you'll most likely have strict contamination requirements"). Implementation is then based on results from analysis. Control structure typically lends itself to implementation in a set of rules. Conceptual knowledge, especially deep knowledge, is more difficult to represent in rules. Rules express surface knowledge fairly well, but the number of rules required to cover most potential possibilities or behavior that are manifested make rule-based systems difficult to manage and implement for large domains. One alternative planned for our second year activities is use of model-based reasoning; specifically causal models.

Another factor that has made this activity successful is the profile of knowledge engineers. Knowledge engineers are from a testing environment and therefore, have some understanding of the domain. This is significant because knowledge acquisition becomes a little easier. Since they already understand or speak the same language as the experts, communication is simplified and some time is saved in having to establish a common vocabulary. Knowledge engineering tasks are also divided up by pairs. A knowledge engineer who is more fluent in test activities and does well in interpersonal relationships conducts most interviews and performs knowledge analysis for focussing subsequent interviews, clarifying information and providing deduced rules and relationships to the developer. The other half of this team is a strong developer who also does knowledge analysis, but takes rules and relationships deduced and implements them in a working prototype. Both knowledge engineers, however, must work closely together and with experts.

## Prototype Tools Developed

Based on knowledge captured in environmental testing, a prototype expert system advising engineers in one integration and test task was developed to demonstrate feasibility of utilizing captured knowledge in a tool to perform a specific task. The prototype Spacecraft Test Assistant (STA) serves as an assistant to a human, who evaluates the expert system's recommendations and acts based on confidence in those recommendations. By capturing existing specific knowledge, an "on-line expert" was available to assist less experienced engineers in testing well defined aspects of spacecraft and large instruments, thus freeing experts to solve more complex and anomalous problems.

STA is a rule-based system implemented in a commercial PC-based expert system shell. Rules deduced from environmental testing allow STA to provide a preliminary list of types of tests to conduct, how to perform these tests, and some diagnostic help in the event of failure. With online capabilities, a test engineer can determine needed environmental tests in less time than traditional manual methods. For example, if a spacecraft has an imaging instrument, STA infers with high probability that this instrument contains a charged coupled device (CCD) and therefore, has strict contamination requirements. If a user wants rationale for why this decision was reached, clicking on an explanation "button" will produce a window explaining that a CCD functions best at cold temperatures and therefore, acts as a cold trap for contamination which would impair its capabilities. Consequently, strict contamination requirements are needed to prevent this. Based on strict contamination requirements and other factors, STA would suggest types of thermal vacuum tests that must be done such as bake-out tests, which chamber to use, how to tailor this chamber to meet strict contamination requirements, how to run a bake-out test, where best to place sensors such a residual gas analyzer and how to measure for contamination. The following is a typical query screen for STA:



By Irene Wong Woerner
x4-5396, Section 374

Being Evaluated: Doo Dah Spacecraft

Confidence Threshold: 50 %

Types of Instruments are:

☒ Imaging

☒ Magnetometer

□ Radar

Another prototype developed was Browser. Browser is a hypermedia application implemented in a commercial PC-based tool. Although not a knowledge-based system per se, links created in Browser are very knowledge intensive. Based on knowledge presentation diagrams depicting decomposition of flight system test engineering into logical modular units, raw data is organized into hypermedia for easier browsing. Since relationships between information and data sets are very important, a hypermedia tool provides more insight than, for example, a hard-bound text book. A good source for structural information was existing technical documentation. Knowledge, especially process knowledge, is often embedded in documentation like management plans. For example, a test engineering management plan on a particular spacecraft integration and test program provided a good basis for a test taxonomy for spacecraft test and a source for determining the framework of how decomposition should occur. Often existing documentation can describe a "classic" model of processes. However, real world usually deviates from this model, and experts are required to provide the richness of knowledge that is required to tailor or understand the processes that actually happen. Much of this richness is in the form of heuristics derived from years of experience and lessons learned. There are examples that can be recounted of anomalies that were encountered that may require a change in the basic model. Design of knowledge-based tools should consider ease of changes an integral part of design, because changes can occur even at the most fundamental model level after prototype implementation has already occurred.

An important factor to consider before implementing a hypermedia tool is availability of resources to define and implement relational links. Cost-effective hypermedia systems will become more of a reality if more automated authoring tools are available, because the current method of establishing links is very labor intensive.

## Future Activities

Some activities planned for the second year include interfacing STA and Browser. If a user is unsure of what a query is asking or wants even more information on why a decision was reached, they can browse through a large set of related data before continuing in STA.

Another capability is coupling these tools with commercial data acquisition systems to actually run a test and perform knowledge-based analysis on results. Several micro-based data acquisitions are currently being evaluated.

A major technical objective for the second year is integrating another subdomain, Spacecraft Assembly Facility testing, with the existing knowledge base and creating a systematic process for expansion of scope. Another area that requires work is conflict resolution. Currently, the scheme is to provide conflicting information separately and to indicate the associated expert. However, a technique we will try is interviewing several experts together to resolve conflict. Another technique we will try is recording interviews with video and audio media. We are hoping to discover insight and nuances provided through body language and expression. How this will be transcribed and searched is still an open issue. Video media may, in fact, be more cost-effective than micro-audio cassettes because of large availability.

Potential benefits from this effort are vast, because both its products, such as captured knowledge base and tools, and its processes, such as application of multiple technologies to a specific problem domain, are useful and have broad applications. First, captured expertise on flight system test and integration is irreplaceable and is now in a usable form. Furthermore, retaining this extensive experience base assures continued excellence in JPL's test programs by securing existing knowledge and evolving this knowledge base as new test engineering concepts are acquired.

*The research described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.*

*Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government or the Jet Propulsion Laboratory, California Institute of Technology.*

## References

Hayward, S.A. et al. "Structured Analysis of Knowledge." Knowledge Acquisition Tools for Expert Systems. Boose, J.H. and Gaines, B.R. (editors), Academic Press Limited, 1988.

Kearney, Michele. "Making Knowledge Engineering Productive." AI Expert. July 1990.

Monk, A. F., Walsh, P. and Dix, A.J. "A Comparison of Hypertext, Scrolling and Unfolding as Mechanisms for Program Browsing." People and Computers IV. Cambridge University Press, Cambridge, 1988.

Nelson, T.H. "Getting It Out of Our System." Information Retrieval: A Critical Review. G. Schechter, ed., Thompson Books, Washington D.C., 1967.

Salton, G. and Buckley, C. "On the Automatic Generation of Content Links in Hypertext." Technical Report. Department of Computer Science, Cornell University, Ithaca, New York, April 1989.

Simpson, A. "Navigation in Hypertext: Design Issues." Proceedings from the 13th Annual International Online Information Meeting, London, England, December 1989.

Stachowitz, Rolf A. and Combs, Jacqueline B. "Validation of Expert Systems." Proceedings of the Twentieth Annual Hawaii International Conference on System Sciences, 1987.

Vassiliou, Yannis, et al. "Expert Systems for Business Applications - A Research Project at New York University." Database Engineering, Vol. 6, #4, Dec. 1983.

Wong, Irene. "Application of Expert System and Hypertext Technologies to Test Document Generation: An Update." Proceedings of the Seventh International Test Conference on Testing Computer Software, June 1990.

# *Special Topics*

# A PROVEN KNOWLEDGE-BASED APPROACH TO PRIORITIZING PROCESS INFORMATION[a]

Daniel R. Corsberg
Idaho National Engineering Laboratory
P.O. Box 1625
Idaho Falls, ID 83415

Many space-related processes are highly complex systems subject to sudden, major transients. In any complex process control system, a critical aspect is rapid analysis of the changing process information. During a disturbance, this task can overwhelm humans as well as computers. Humans deal with this by applying heuristics in determining significant information. This paper describes a simple, knowledge-based approach to prioritizing information. The approach models those heuristics that humans would use in similar circumstances.

The approach described in the paper has received two patents and has been implemented in the Alarm Filtering System (AFS) at the Idaho National Engineering Laboratory (INEL). AFS was first developed for application in a nuclear reactor control room. It has since been used in chemical processing applications, where it has had a significant impact on control room environments. The approach uses knowledge-based heuristics to analyze data from process instrumentation and respond to that data according to knowledge encapsulated in objects and rules. While AFS cannot perform the complete diagnosis and control task, it has proven to be extremely effective at filtering and prioritizing information. AFS has been used for over two years as a first level of analysis for human diagnosticians. Given the approach's proven track record in a wide variety of practical applications, it should be useful in both ground- and space-based systems.

## INTRODUCTION

The first section of this paper discusses traditional systems and processes where alarms and information overload have been a problem. Examples of current and future space-related systems with similar characteristics are also described. The following sections discuss how the information overload problem has been addressed in the past and what makes the Alarm Filtering System approach both unique and practical. We then provide a detailed description of the approach as well as a discussion of current applications. In the final section, we look at how the approach might be used today and how future diagnostic, management, and control systems might use the technology.

[a] Work supported by the Department of Energy under DOE Contract No. DE-AC07-76ID01570. The Department of Energy has been granted patents on the approach implemented in the Alarm Filtering System (AFS) and on the application of AFS at the Advanced Test Reactor (ATR).

## THE GENERIC PROBLEM OF INFORMATION OVERLOAD

Today's process and system operators are presented with an ever increasing number and complexity of alarms and information displays. While operators can assimilate this information, stress is placed on the operator in doing so. This stress can lead to mistakes. Pertinent information must be presented in a manner that is clear and concise. Unfortunately, new data acquisition and display technologies have exacerbated the problem rather than alleviated it. The tendency has been to instrument (and alarm) virtually every measurable system parameter. The problem is further compounded by funneling all the measurements and alarms into the control environment. The beleaguered operators are left to analyze massive amounts of data.

This information overload scenario is common across the entire spectrum of process operations and control. The many problems inherent to nuclear power plant control rooms are well documented (Banks, 1981, Christie, 1982, Wahlstrom, 1983). The cascading effect of one or two causal events in a reactor plant can activate hundreds of alarms during the initial five seconds of a transient (Felkel, 1984). While research had begun prior to the Three Mile Island (TMI) accident, that event acted as a catalyst for a generation of work in the area of information display and human factors.

Other industries and types of systems suffer from similar problems. Telecommunications networks can be brought to their knees by a single event that propagates throughout the system. The dynamic nature of many chemical processes (oil refineries, off-gas and scrubber systems, toxic chemical disposal facilities, etc.) leaves operators with little time and no margin for error. Other less dynamic processes can generate a steady stream of alarms and information that gradually wear an operator down. Operators may be less alert when a real problem arises.

Space-related systems based on the ground, in vehicles and platforms, and on the lunar surface will be extremely dynamic and complex. The push towards autonomous control will leave fewer human operators to deal with more problems on a larger number of diverse systems and processes. These systems will range from ground-based systems such as launch and communications to space systems such as propulsion, power, environmental, and operational. Indeed, the importance of recognizing and responding to casualties has already been recognized for future missions and operations. In discussing the control of transfer and orbital operations, Ramsthaler states, "the first line of defense against a major failure is adequate knowledge of the situation as it is developing" (Ramsthaler, 1988). In discussing potential problems related to propulsion, Ramsthaler makes the point that "in order to avoid major damage to an engine, action would have to be taken within seconds of the failure for many of the items." These concerns will continue well into tne future as the same types of systems are specifically mentioned in proposals for projects that would culminate 50 years from today (West, 1989).

## GENERIC APPROACHES TO SOLVING INFORMATION OVERLOAD

One primary goal of data acquisition and information display is to present the needed information at the appropriate time. An effective way of addressing information overload would be during the design phase of the system or process to be monitored. By properly identifying the required information, system designers could reduce the incoming data to a manageable amount. Unfortunately, one can postulate situations where virtually any specific process information could be useful. As a result, everything is instrumented, and all of that data is brought into the control environment.

Bringing all the data into the control environment ensures the needed information is always available. In fact, it ensures that for a given situation a large amount of extraneous information is also always available. This extraneous information overloads the operators. Measures must be taken to focus operator attention on important information as effectively as possible. Various types and levels of information processing have been used for decades (Baker, 1985). The functional grouping of information and alarms is one rudimentary form. With the advent of computers and microprocessor based displays, static prioritization has become feasible. Static prioritization, which is widely available in industry today, allows facilities to assign a predetermined priority level to an alarm or piece of information. That priority remains constant (or static) no matter what the situation is. Recall the goal of providing the needed information at the appropriate time. Static prioritization is a step in that direction. It does not, however, adequately account for the dynamic nature of processes and systems. When looking at the entire spectrum of process states, a piece of information probably does not have a single level of importance relative to other information from the process. Often, that relative importance varies from a high level to a level of being extraneous.

The next step in information and alarm prioritization has to take into account the state of the process or system being monitored. The Alarm Filtering System (AFS) is such a step towards providing needed information at the appropriate time.

## ALARM FILTERING SYSTEM

As mentioned earlier, considerable effort has been made in developing operator aids for the nuclear power industry. Several tools have been proposed or implemented in such systems as DMA (Diagnosis of Multiple Alarms) (Danchak, 1982), STAR (Felkel, 1984), and DASS (Disturbance Analysis and Surveillance System) (Long, 1980). These systems addressed the dynamic nature of processes by using logic or cause-consequence trees to identify the process state and emphasize information accordingly. These trees are difficult and expensive to build, tend to be inflexible to change, and are not easily maintained over the life of the plant (Baker, 1985).

The Alarm Filtering System (AFS) was originally developed to address the problems caused by onrushes of alarms in nuclear power control rooms. We have since applied the approach to other processes and to other information besides just alarms. We have found the approach to be responsive (in terms of processing information), relatively easy to develop and maintain, and effective in helping to manage the information in the control environment.

AFS determines the importance of alarms and information relative to its knowledge of the current plant state. When process information is provided, AFS assigns a level of importance to it. As the process subsequently changes and other related information becomes available, that level of importance is reevaluated and possibly changed. Thus, the prioritization is dynamic, changing as the process changes. Important information is emphasized while information not pertinent to the current situation is deemphasized and, in some case, eliminated. AFS uses relationships between alarms and information as a basis for determining importance. These relationships and their associated rules are used by AFS to:

- Generate a description of a situation implied by combinations or sequences of information,

- Suppress information that simply confirms or is a direct result of a previously described situation,

- Emphasize information that does not correlate with previous conclusions or information that is expected (due to previous conditions) but is not received within specified time limits. This expected information is typically the result of automatic system response to a process state or operator action.

The approach used in AFS resulted from applying expert system concepts to the problem. We looked at the information processing problem from the operator's view point and modelled the operator's methodology for rapidly analyzing changing information. We recognized that operators use relationships between pieces of information as a basis for determining relative importance. Five types of relationships were identified during the development and application of AFS. Each type of relationship has a set of possible responses and decisions that can be made. Thus, each type has a set of rules associated with it that model how alarms and information should behave and what levels of importance should be assigned. The five types of relationships are level and direct precursors, required actions, first-out, and blocking conditions. These are discussed in greater detail in the descriptions of the two patents on the approach (Corsberg, 1988, 1989). There is nothing particularly complex about these relationships. It is their practical application to process information management that produces effective results. The level precursor relationship is an excellent example. This relationship typically occurs when the same parameter has two alarm setpoints. Thus, alarm A's setpoint would be at one level (a lower reading), while alarm B's setpoint would be at another (higher) reading. Alarm A should always be activated before B. If both are activated, then B should deactivate prior to A. In terms of prioritization, if both A and B are activated, B should be at a higher level of priority than A. If only A is activated, it should be emphasized at the highest level (unless another related piece of information affects its priority). As mentioned above, these priorities are dynamic. Thus, if both A and B were activated, and B is then deactivated, A's priority would be updated based on the new set of information.

The use of objects and their associated mechanisms has been of particular importance in the successful development of AFS. AFS uses information about the processes and systems being monitored as well as parameters and other types of information. Each of these entity types is represented by a class that acts as a blueprint for building specific objects. All objects in a class will have the same structure because they were all built from that same blueprint. Each specific object in a class has a different information content based on the entity it represents (rather than on the type of entity).

In addition to providing structure for objects, classes also associate functionality with the objects. This functionality takes the form of programming procedures that are invoked when objects send messages to each other. These procedures are common to all objects in a particular class. The rules concerning behavior and priority levels are encapsulated in these procedures. The ability to associate a procedure with an entire class of objects has greatly increased the modularity of AFS and reduced the total number of rules required. Specific data about each piece of information is contained in the object representing that piece of information. The more general knowledge about responses and actions caused by the relationships is contained in the procedures. This separation of knowledge provides the modularity to make AFS flexible. During the development of an implementation, developers can focus on the process and its parameters rather than on the more generic portions of the approach. Changes can be made with minimal impact on nonrelated portions of the implementation.

## AFS APPLICATION AND KNOWLEDGE ACQUISITION

AFS has been installed in nuclear reactor control room simulator and in the control room of a chemical processing facility. Two important issues in the application process are the information displays and the knowledge acquisition process. As the reader may have noted already, this paper has not mentioned information displays. One reason is that AFS is relatively independent of these displays. AFS acts as an in-line processor between the instrumentation and the information displays in the control environment. AFS simply assigns a priority to each piece of information. What is done in response to that priority is entirely up to the specific facility (and, possibly, even a specific operator). AFS has been used in facilities where the associated display is a graphical mimic of the process. The same AFS approach has been used in conjunction with simple scrolling text windows where the AFS-assigned priorities were used to determine what to scroll off the window and what to leave on. This modularity allows AFS to be integrated into any environment that has an open architecture.

When discussing AFS, the most common and immediate question is about how the relationships between the information are determined. The relationships are defined through the process of knowledge engineering, a discipline that has grown out of expert systems and artificial intelligence. We start by clearly defining the information and alarms to be processed. (This turns out to be a difficult task in itself.) We use engineering, training, and operational documentation as a way of becoming familiar with the process and its terminology. We then go through an exhaustive and iterative process of identifying and justifying relationships between information and alarms. This process of extracting the knowledge, putting it to paper, and then refining it had little to do with computers. In fact, the close examination of data acquisition and alarm systems revealed several items and assumptions that were incorrect.

During the design of a facility, engineers will anticipate some of the information problems and will effectively tune them out. However, not everything can be predicted, and there are always extraneous, spurious bursts of information that are generated once a process or system begins operation. All AFS applications have had significant operating experience to draw on. This experiential knowledge has been the foundation of the knowledge bases in these applications. Many potential AFS space applications would not necessarily have this experiential knowledge to draw on. Potential areas to acquire this knowledge from include training facilities and simulators. The fact that AFS can be changed easily would also allow for modification once systems have been deployed. These changes could be effected remotely since AFS is implemented entirely in software.

AFS has proven to be an effective aid to operators. It adds minimal time to transmitting process information to the control environment. Versions implemented in the programming language LISP add between 10 and 20 milliseconds to the total time response. As implementations in more traditional programming languages such as C come online, these times will be further reduced. One application has been used for nearly three years without a single failure. That application has reduced message traffic by more than 80%.

## AFS AND SPACE

AFS is a step towards more effective management of process information. It is not a stand-alone system. AFS applications must be developed and integrated into a total environment of information acquisition, processing, and display. Many potential space-related applications could be developed today. These include ground-based launch, control, and communication systems where information management problems have been identified. In these cases, AFS could be used to directly influence the information being displayed to operations personnel.

Future applications would probably use the AFS approach in a different way. AFS uses experiential knowledge and has no real in-depth understanding of the systems and processes being monitored. As more sophisticated diagnostic and control systems are developed, AFS's role would change from that of providing more effective information to humans. Instead, AFS would act as a front-end to automated software with complicated models and knowledge bases. The knowledge used in AFS is a highly effective way of rapidly identifying a situation and can be envisioned as handling a high percentage of information management problems. To handle more difficult problems, AFS applications would need to be incorporated into a much larger environment of cooperating knowledge-based systems. These other systems would handle the more complex and computationally intensive tasks of diagnosis and control. AFS applications would still provide the needed information at the appropriate time. Rather than providing that information to humans, AFS would be working with other software systems, improving their quality and effectiveness.

## CONCLUSION

This paper has described a knowledge based approach to prioritizing process information. This approach, AFS, has been developed and successfully used in nonspace applications. AFS has proven to be an effective step towards solving many real-time information management problems in control rooms. These same problems exist, or will exist, in the control environments of current and future space-related systems. As such, AFS should be generally applicable to a wide variety of space applications.

## REFERENCES

Baker, et al (1985). *An Experimental Comparison of Three Computer-Based Alarm Systems: Results and Conclusions. HWR-142, OECD Halden Reactor Project.*

Banks, W. W., and Boone, M. P. (September 1981). *Nuclear Control Room Annunciators: Problems and Recommendations.* NUREG/CR-2247, EGG-2103.

Christie, A. M. (October 1982). *DICON - An Expert Systems Approach to Diagnostics and Control Guidance with LMFBR Application.* WARD-SR-94000-37.

Corsberg, D. R. (1988, June 7). *Functional Relationship Based Alarm Processing.* United States Patent No. 4,749,985.

Corsberg, D. R. (1989, March 14). *Functional Relationship Based Alarm Processing System.* United States Patent No. 4,812,819.

Danchak, M. M. (September, 1982). *Alarms Within Advanced Display Streams: Alternatives and Performance Measures.* NUREG/CR-2276, EGG-2202.

Felkel, L. (1984). *The STAR Concept, Systems to Assist the Operator During Abnormal Events.* Atomkernenergie, Kertechnik, 45(4).

Long, A. B., et al (December 1980). *Summary and Evaluation of Scoping and Feasibility Studies for Disturbance Analysis and Surveillance Systems (DASS).* Topical Report EPRI NP-1684.

Ramsthaler, J. H., et al (October 1988). *Safe, Compact, Nuclear Propulsion (Solid Core Nuclear Propulsion Concept).* AFAL-TR-88-033, EGG-ES-8093.

Wahlstrom, B. (1983, May 23-28). *The MMiF For Alarm Presentation. Enlarged Halden Programme Group Meeting on Computerized Man Machine Communication.* Leon, Norway.

West, C. (1989). *Project Longshot: A Mission to Alpha Centauri.* United States Naval Academy, Annapolis, Maryland.

# DESIGN OF AN INTELLIGENT INFORMATION SYSTEM FOR IN-FLIGHT EMERGENCY ASSISTANCE

Stefan Feyock
Stamos Karamouzis

Computer Science Department
College of William & Mary
Williamsburg, VA 23185

## Abstract

The present research has as its goal the development of AI tools to help flight crews cope with in-flight malfunctions. The relevant tasks in such situations include diagnosis, prognosis and recovery plan generation. Investigation of the information requirements of these tasks has shown that the determination of paths figures largely: what components or systems are connected to what others, how they are connected, whether connections satisfying certain criteria exist, and a number of related queries. The formulation of such queries frequently requires capabilities of the second-order predicate calculus. We describe an information system that features second-order logic capabilities, and is oriented toward efficient formulation and execution of such queries.

## 1. Introduction

The research described in this report was performed in conjunction with the Intelligent Cockpit Aids (ICAT) project conducted by the Vehicle Operations Branch at NASA Langley Research Center. The goal of this project is to develop artificial intelligence (AI) techniques and systems to assist flight crews in the performance of their tasks. Such assistance can become particularly crucial when malfunctions occur; a significant portion of the project is accordingly devoted to the development of tools that will help flight crews cope with in-flight faults.

The design possibilities for such software tools range from passive systems to be used as information resource by the flight crew, through systems that autonomously determine and suggest appropriate actions, to software that takes complete control of the aircraft in case of emergency*. We begin by considering the low-autonomy end of this spectrum, with systems that confine themselves to producing answers to queries posed by the flight crew. Once the functions of such systems are understood, we can consider what aspects of the system's *user* can be automated.

The questions that arise are:

1. what type of information is likely to be required in case of in-flight malfunctions, and

* The set of pilots who consider the latter class of systems to be a good idea is essentially null; nonetheless, certain recently developed aircraft types exhibit a disquieting degree of autonomy in deciding how to reconfigure themselves after a malfunction.

2. how is it to be produced?

As regards item (1), it is clear that the most critical issues to be determined are:

1. diagnosis: what is the nature of the fault?

2. prognosis: how will this fault affect the subsequent operation of the aircraft, in terms of flight characteristics as well as fault propagation to additional components?

3. recovery planning: what is the appropriate response to the malfunction?

Ongoing research by members of the ICAT group has resulted in a number of sophisticated AI tools appropriate to the task of producing in-flight diagnoses [1,7]. The research described in this paper has concentrated on the production of *prognoses*, given a diagnosis produced by such diagnostic systems. In the course of our work we have come to the following conclusions:

1. traditional information (i.e. database) systems technology is inadequate for the task of prognosis. For example, we have found that the underlying database system must be able to respond to queries of the type "is there an electrical path between components A and B? A hydraulic connection? Any sort of connection? Such queries involve quantification over paths and relations, and thus belong to the realm of the second-order predicate calculus, beyond the capabilities of traditional database systems.

2. Standard rule-based expert systems also fail to meet the requirements of the task. The rules within such systems are to a large extent distillations of experts' responses to familiar problem situations. Malfunctions by their very nature create chaotic conditions rife with unforeseen consequences that may interact in unexpected ways. The brittleness of rule-based systems in the face of such situations is well-known; a deeper kind of reasoning is required to generate adequate responses.

Our approach to these problems has been to embed a variety of models in our information system to allow deep reasoning to take place, and to develop a database system capable of the second-order operations that occur frequently in the course of such reasoning. Fig. 1 depicts the overall organization of the resulting system.

As can be seen, the model-based information system (MBIS) consists of a number of submodels of a quite diverse nature, coordinated by an entity which may be an in-the-loop human, or may be an "automated flight engineer" that assumes the human user's functions to the extent feasible. The LIMAP utility is the second-order database discussed above, while the semantic net serves as a central information resource helping to tie the submodels of MBIS together. In this paper we describe the LIMAP submodule of the MBIS system, and how it interacts with the other components to fulfill its role.

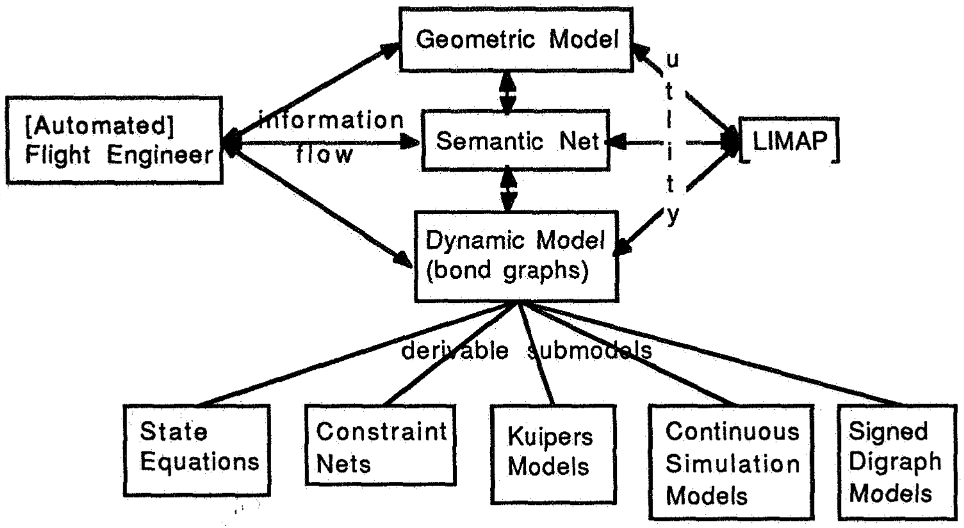


Fig. 1 The Model-Based Information System

## 2. Second-Order System: Rationale

We will introduce LIMAP by describing a typical situation in which the need for a second-order database system arises. As indicated above, ICAT researchers have developed a number of diagnostic tools to assist the flight crew with the task of determining the cause of in-flight malfunctions. One of the most important of these is the DRAPHYS system [1], a model-based reasoner that determines which components' malfunction best accounts for the observed symptoms. DRAPHYS uses a digraph model of an aircraft system, with nodes representing primitive components*, and the arrows connecting nodes representing functional and physical dependencies. Component B is said to be *functionally dependent* on component A if the proper functioning of B depends on the proper functioning of A. Component B is *physically dependent* on component A if damage to A can propagate through space to component B. For example, the control surfaces of an aircraft are functionally dependent on the hydraulic system, since they will cease operating if the latter fails. On the other hand, if a hydraulic line can be severed by a disintegrating turbine, the line is physically dependent on the turbine. While functional dependencies can generally be determined by considering causal relationships in the physical system, physical propagation is typically the result of leakage of some substance or form of energy (e.g. as in an explosion), and is inherently unpredictable. Reference [5] contains a discussion of model-based reasoning about physical fault propagation.

We will illustrate the concepts involved by means of an example: a jet engine. Fig. 2 show a schematic of a dual-fan jet engine, while Fig. 3 gives the functional dependency graph for this engine. (we ignore physical dependencies for the sake of simplicity)

* A component is deemed to be *primitive* if it is considered to be atomic, i.e. to have no subcomponents, with respect to the model. This property clearly depends on the granularity of the model; thus, an entire engine could be treated as primitive if we are content with the diagnosis "engine malfunction" rather than, say, "compressor stall".

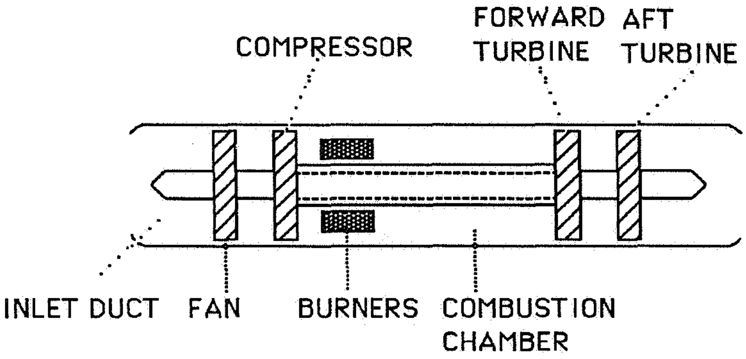


Fig. 2 Jet Engine: Schema

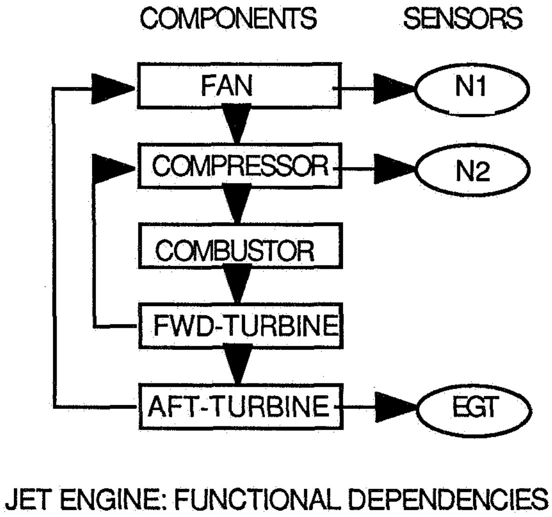


Fig. 3 Functional Dependency Model of Jet Engine

When a malfunction occurs, the DRAPHYS model-based diagnostic system extracts information from this model by means of the following procedure:

```
Suspect every possible component*;
Use model to determine consequences of each failure;
Eliminate suspects inconsistent with model predictions;
```

More precisely:

```
for each C in SET-OF-PRIMITIVE COMPONENTS do
        if "C has failed" is a valid hypothesis
        then add C to SET-OF-VALID-HYPOTHESES;
end for;
```

* Heuristics are used to prune the set of "possible" components.

With luck, SET-OF-VALID-HYPOTHESES will contain only one element. If it contains more, DRAPHYS waits for more symptoms to develop and disambiguate the diagnosis.

We must specify how it is determined that "C has failed" is a valid hypothesis (possible diagnosis). Here is an informal description of how this is done:

```
Primitive component C is a valid hypothesis iff
        there is a POSSIBLE PROPAGATION PATH from C
          to every symptomatic sensor
```

A path is a POSSIBLE PROPAGATION PATH iff every instrumented component on the path has at least one symptomatic sensor

Here is a predicate calculus formulation of this stipulation:

```
valid-hypothesis(component)  <=>

is-primitive(component)  &
(A s) {is-sensor(s) & is-symptomatic(s) =>
(E p) [(path(p, /*from*/ component, /*to*/ s) &
(A n)[is-node(n, /*on path*/ p) => not is-ok(n) or is-unknown(n)]]}

is-ok(n) <=> (A s)[is-sensor(s) & instruments(s,n) =>
        not  is-symptomatic(s)]

is-unknown(n)  <=>
        not  (E s)[is-sensor(s)  &  instruments(s,n)]
```

/* instruments(s,n) is true if s is a sensor measuring some attribute of n */

This definition contains the construct "does there exist a path from x to y such that every node on that graph fulfills certain constraints?". The node constraint in this case is that every instrumented node on the path must be symptomatic. The wff states that a primitive component C is a valid hypothesis iff there is such a path from C to every symptomatic sensor. Such quantifications are difficult to express in the first-order predicate calculus; if each edge of a graph can denote a distinct arbitrary relation, a second-order formulation is required, since it must contain (among other things), a passage such as this: "...for all nodes m and n on path p there exists relation R such that R(m,n)...".

Queries of the form "is there a relation R such that nodes a and b are in relation R? "is there a path from x to y? a path fulfilling constraint C? where can I go from x? how can I get to x?" arise frequently both in AI and out of it. Such queries, which involve quantification over relations, correspond to statements in the second order PC. The question is: how can such queries be implemented efficiently? In the following section we describe the structure and implementation of an information system appropriate to the requirements we have outlined.

## 3. LIMAP

The second-order operations that have been discussed raise the question of whether existing languages, particularly Prolog [3], already fulfill all the stated requirements. Some Prolog implementations do allow predicates to be variables, permitting at least existential quantification. Furthermore, it is straightforward to construct Prolog procedures that determine the possible paths between nodes in an arbitrary network, as well as allowing the user to stipulate constraints on these paths, to browse through them, and to perform a variety of similar operations. The problem is that such capabilities can easily become prohibitively inefficient. This is the motivation for the creation of LIMAP, which offers a number of Prolog-like capabilities, but whose implementation is oriented toward storage and execution efficiency.

Most AI search/representation techniques are oriented toward a potentially infinite domain of objects and arbitrary relations among them. Experience has shown that in practice much of what needs to be represented in AI can be expressed using a finite domain and unary or binary predicates. Even in cases where large relations are involved, they are almost invariably sparse as well. A major subassembly of an aircraft, for example, may have thousands of subcomponents, but the relations of interest among them, such as IS-A, PART-OF, functional dependency, etc., each involve only a small proportion of the full set of components.

As regards the arity of predicates, the most widespread and useful representations of AI are semantic nets, frames, schemas, inheritance hierarchies, and related constructs. All of these can be expressed by means of predicates of at most two arguments*.

### 3.1. Representations of Predicates

Since most of the relations of interest will be sparse and have no more than two arguments, specialized techniques for representing and manipulating such relations may be used. Unary predicates over a finite fixed domain are well represented by bit strips (boolean vectors), while adjacency matrices (AM's; see Appendix) are appropriate for binary predicates. A representation based on (optionally sparse) vectors and adjacency matrices accordingly forms the basis of the LIMAP implementation.

By way of example, Fig. 4 depicts the adjacency matrix representing the jet engine functional dependency predicate** FD(x,y) of Fig. 3 over the domain D = {fan, compressor, combustor, fwd-turbine, aft-turbine, N1-sensor, N2-sensor, EGT-sensor}. By definition of adjacency matrices, a "1" in row i, column j denotes an arrow from node i to node j.

Boolean vectors are equally useful for representing unary predicates. The vector IS-SENSOR = <0, 0, 0, 0, 0, 1, 1, 1>, for example, represents the IS-SENSOR predicate; a "1" in position i denotes the fact that element i is a sensor.

* There are, of course, well-known techniques for expressing any n-ary (n > 2) predicate in terms of binary predicates. Most knowledge can be expressed using unary and binary predicates without performing such a decomposition.

** The notation is somewhat misleading: FD(x,y) iff x *functionally determines* y, i.e. iff y is functionally dependent on x.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1. fan | | 1 | | | | 1 | | |
| 2 compressor | | | 1 | | | | 1 | |
| 3 combustor | | | | 1 | | | | |
| 4 fwd-turbine | | 1 | | | 1 | | | |
| 5 aft-turbine | 1 | | | | | | | 1 |
| 6 N1 sensor | | | | | | | | |
| 7 N2 sensor | | | | | | | | |
| 8 EGT sensor | | | | | | | | |

Fig. 4

Given an AM representation, many useful second-order operations can be expressed concisely and efficiently:

(EXISTS X) X(a,b)?
(FORALL X) X(a,b)?
(EXISTS P) P a path from node a to node b?

Such queries correspond to finding parents, siblings, descendants, routes between nodes, etc. For example, we can obtain the set of instrumented components simply by performing the boolean matrix multiplication FD x IS-SENSOR$^T$. Similarly, the question of whether a path exists between, say, the fan and the EGT sensor is trivially answered by noting whether FD$^*$[1,8] contains a 1.

It is frequently necessary, however, to determine not only the existence of a path, but the path itself, as ordered set of nodes. In addition, if the problem representation requires a large number of predicates over D or D x D, maintaining a separate AM for each predicate becomes unwieldy.

A straightforward extension of the adjacency matrix representation allows digraphs with labeled edges to be represented. Under this extension, the elements of the AM are sets of labels, rather than 0 or 1. Element ij contains label P iff the semantic net contains an arrow from $d_i$ to $d_j$ representing (labeled with) predicate P; the empty set denotes the absence of an edge. Such extended AMs are termed *symbolic adjacency matrices* (SAMs).

It is worth noting that SAMs provide a mechanism for implementing second-order queries. The wff (EXISTS X) X(a,b), for example, can be decided simply by determining if row a, column b is the empty set.

### 3.2. PSAMs

A straightforward extension of Warshall's Algorithm to symbolic adjacency matrices allows efficient computation of a *path symbolic adjacency matrix* (PSAM; see Appendix), a matrix whose ij entry contains the set of all paths from node i to node j. It is this capability that makes quantification over paths feasible.

## 4. The LIMAP language

The current version of LIMAP is an experimental test bed for determining what facilities are required, and how to implement them most efficiently. In this section we describe the language features that have been implemented to date, and plans for future.development. The prototype version of LIMAP is implemented in Common LISP.[8]. A C implementation, oriented toward optimizing the bit level representations and operations on which the efficiency of LIMAP depends, is under way.

### 4.1. The LIMAP DDL/DML

Every language has an underlying implementation model. As we have seen, the LIMAP implementation model is based on a representation that employs boolean and symbolic vectors and adjacency matrices to represent unary and binary predicates, as well as an efficient transitive closure computation capability that allows boolean or symbolic path matrices to be computed and manipulated.

As is the case for an ordinary first-order database system, LIMAP capabilities are invoked via a language interface that consists of two parts. One is the data definition language (DDL) for specifying both the data the system is to contain as well as "metadata", i.e. information about the structure and constraints that govern the data contained in the system. The other is the data manipulation language (DML), the subset of the language concerned with the specification of queries and updates on the the data. We will categorize.the LIMAP functions accordingly. A brief summary of the LIMAP DDL and DML follow; [4] contains a complete listing.

#### 4.1.1. DDL operations

The basic DDL operations are

```
DEFREL <name> <specification> <type> <representation>
<specification> ::= (<number>) or (<number> <number>)
<type> ::= boolean | symbolic
<representation> ::= sparse | dense
```

and

```
DELREL <name>
```

to define, respectively delete, a relation. DEFREL creates a new array according to the values of the parameters, and binds this array to <name>. <specification> stipulates whether the array will be a vector or matrix, as well as the index range(s). <type> specifies whether the declared relation will be represented by a boolean or symbolic array, whereas <representation> allows the user to choose a sparse or dense array representation. Defaults are provided for the SPECS and REP parameters. The attributes of an array, as well as a pointer to the data structure representing the array elements, are inserted into an internal symbol tables. As might be expected, the effect of DELREL <name> is to unbind <name>, effectively deleting the array.

#### 4.1.2. DML operations

The major DML operations are

| | | | |
|---|---|---|---|
| STORE | relname | value [row] column | Store value |
| RETRIEVE | relname | [row] column | Retrieve contents |
| TCLOSE | relname | | Transitive closure |
| PATHS | relname | row column | All paths |
| MULT | relname | relname | Multiply |
| TRANSPOSE | relname | relname | Transpose |

STORE and RETRIEVE perform the indicated operation on the specified array position, in accordance with the array's type and representation, while MULT and TRANSPOSE typify a variety of standard matrix operations made available by LIMAP. Except in DEFREL it is transparent to the user whether the array representation is sparse or dense. This transparency extends to the other attributes of the array wherever possible.

### 4.1.3. Path Operations

The TCLOSE and PATHS operations form the core of LIMAP's path manipulation capability. TCLOSE computes the transitive closure or the PSAM of the indicated array, depending on its type. A pointer to the resulting closure array is stored in the symbol table for relname, and may henceforth be accessed by queries referencing paths. In particular, PATHS[i j] retrieves the set of all paths from node i to node j, enabling quantification over paths. The LIMAP implementation of DRAPHYS, shown below, contains an example of this capability.

## 4.2. Control structures

The distinction between procedural and non-procedural predicate calculus specifications blurs if the underlying domain is finite, since the FORALL and EXISTS quantifiers map in an obvious way to loops ranging over the domain elements. It has been our goal to give the LIMAP DML as non-procedural a character as possible. In particular, LIMAP notation is an adaptation of the (function-less) predicate calculus, with extensions to allow data retrieval in addition to data specification*, .Perhaps surprisingly, we have found that minimal modifications of the control macros described in [2] were suitable for the task of expressing the required quantifications. Here is a summary of the general form of the control structure implemented by these macros:

(FOR ((<variable$_1$> :IN <set$_1$>)

    (<variable$_n$> :IN <set$_n$>) )
    [:WHEN <when-expression>]
    <FOR-keyword> <expression$_1$> ... <expression$_n$> )

The construct (<variable$_i$> :IN <set$_i$>) causes the variable to iterate over the elements of the set, which may be specified as a list, a vector, or a matrix row or column. Unless a false when-expression is present, the FOR-body is evaluated and a result is produced as governed by the FOR-keyword. Iteration then proceeds to the next set of variable values.

FOR-keywords

| | |
|---|---|
| :ALWAYS | true if all the values of body are true |
| :FILTER | produce a list of the non-NIL values of body |
| :FIRST | produce the first non-NIL value of body |
| :SAVE | produce a list of all values of body |

While the description of these constructs is procedural in form, the effect when programming in this notation is that of writing FORALLs and EXISTs, with the proviso that any variable values that are found to "EXIST" are col-

* For example, a "yes" answer to (EXISTS X)(FORALL Y)P(X,Y) is insufficient; the actual X-value must be retrieved.

lected in accordance with the FOR-keyword and returned as value. The following section contains an example application of LIMAP: a LIMAP specification of DRAPHYS.

## 5. DRAPHYS in LIMAP

The operation of DRAPHYS has been described previously, both in informal terms and by means of predicate calculus wffs. Here is a LIMAP version; since the notation bears strong analogies to the predicate calculus specification, we present it without further explanation.

```
; DRAPHYS in LIMAP

; The list COMPONENTS contains all engine components, including sensors
;
(defun  determine-hypotheses  (components  symptomatic-sensors)
; components =def set of all components to be considered as hypotheses
(for (c :in components) :when (is-valid-hypothesis c) :filter c)
)

(defun  is-valid-hypothesis  (c  symptomatic-sensors)
(for (s :in symptomatic-sensors) :always (exists-bad-path c s) )
)

(defun  exists-bad-path  (c  s)
 (for (p :in (paths 'engine c s) ) ; paths from c to s
      :first (for (c :in p) :always (not-known-ok c) )
))

(defun  not-known-ok  (c)
  (or (null (instrumentation c)) (symptomatic c))
)  ; symptomatic is a  boolean vector

(defun instrumentation (c)   ;   returns list of sensors associated with c
(for (s :in components)
:when (and (is-sensor s) (retrieve 'engine c s)) :save s)
)
```

## 6. Conclusion

We have described a programming system oriented toward efficient information manipulation over fixed finite domains, and quantification over paths and predicates. The initial motivation for the creation of such a system was the fact that the need for such operations arose frequently in the diagnosis/prognosis generation problem domain. Since then it has become apparent that the facilities provided are useful and applicable over a much wider range of problems, both within and outside of AI.

LIMAP's predicate-oriented DDL and DML are reminiscent of another predicate-oriented language: Prolog. There is, however, an important omission: Prolog contains a built-in inference engine for processing rules, while LIMAP does not. As it happens, since SAM entries can be arbitrary s-expressions, rules are easily added to LIMAP. This is an artifact of the fact that the current implementation language is LISP, and does not generalize to other (planned) implementation vehicles such as C. Addition of a rule capability and inference engine forms a major area of current research, as does optimizing implementation efficiency.

Our experience to date has shown that LIMAP is applicable to a wide range of problems. While LIMAP, if abused, is as capable of inefficient operation as any other misused programming system, we have found that for every problem yet attempted there has existed a LIMAP formulation that was concise, comprehensible, and for which LIMAP's facilities constituted a highly efficient problem representation.

## APPENDIX

We present a brief review of graph-theoretical terminology occurring in the text; see [6] for a detailed discussion.

### Digraphs

A directed graph (digraph) is 2-tuple <N,E>, where N is a finite set of nodes, and E a finite set of edges. An edge is a member <a,b> of NxN. A labeled digraph is a 3-tuple< N,E,L>, where N is as before, L is a finite set of labels, and E is a finite set of labeled edges, with labels in L. A labeled edge (with label in L) <a,l,b> is a member of NxLxN.

It is easy to see that digraphs are a graphic representation of binary predicates over finite domains If P(x,y) is a predicate over domain DxD, then digraph G = <N,E> represents P if P(a,b) iff <a,b> in E.

Whereas an unlabeled digraph can represent a single predicate, labeled digraphs whose label set is a set of predicate names can represent multiple binary predicates over the same domain DxD simply by letting edge <a,p,b> denote the fact that predicate p(a,b) is true; the absence of such an edge denotes that p(a,b) is false. Extending the notation, we allow edges to be labeled with *sets* of predicate names; an edge <a,{$p_1$,...,$p_n$},b> is an abbreviation for the set of edges <a,$p_1$,b>, ... <a,$p_n$,b>. Labeled digraphs thus correspond to the familiar *semantic net* construct of AI.

### Predicate Representations

Given the problem of representing a unary predicate P(x) over a finite domain D of fixed size n, an obvious and familiar solution is to use boolean vectors, a.k.a. bit strips: for any $d_i$ in D, P($d_i$) is true (false) iff the i'th component of the vector representing P is a 1 (0). Boolean operations such as AND, OR, and NOT on predicates over D are then representable by the corresponding operations over bit strips, which are efficient on most computers. Similarly, binary predicates Q(x,y) over D x D can be efficiently represented by *adjacency matrices*, i.e. n x n matrices whose ij element is 1 if Q($d_i$,$d_j$) is true, else 0.

### Symbolic Adjacency Matrices

Boolean adjacency matrices can in principle represent labeled digraphs: a separate matrix is assigned to each label, and represents the subgraph of nodes connected by edges bearing that label. In practice this representation can become unwieldy. The number of different labels may be large, resulting in proliferation of adjacency matrices. Moreover, queries such as "is there any path (regardless of labels) from node a to node b?" require that the matrices for all labels be ORed together. An answer to the follow-up query "what are these paths?" is even more difficult to generate from this representation. Such considerations motivate the adoption of symbolic adjacency matrices (SAMs) as representation for labeled digraphs. Element ij of a SAM is P iff the arrow from $d_i$ to dj in the semantic net has label P, else NIL.

### Warshall's Algorithm

Let G be a digraph; then the transitive closure G* of G is a digraph containing an edge <a,b> iff G contains a *path* (of length 0 or greater) from a to b. Warshall's Algorithm (see [6]) is an efficient method for computing G*, given an adjacency matrix representing the G. Intuitively, the algorithm scans the matrix top to bottom, left to right. If a 1 is encountered, say in row i, column j, then row i is replaced by row i OR row j, and the scan continues from position ij.

A straightforward extension, described in [4], of Warshall's Algorithm to symbolic adjacency matrices produces a matrix, termed the *path symbolic adjacency matrix* (PSAM), whose ij entry contains the set of all paths from node i to node j. It the generation of the PSAM matrix that makes the quantification over paths feasible.

## REFERENCES

[1] Abbott, K., *Robust Fault Diagnosis of Physical Systems in Operation,* Ph.D. Dissertation, Computer Science Department, Rutgers University, New Brunswick, NJ, May 1990.

[2] Charniak, E., et al., *Artificial Intelligence Programming*, 2nd ed., Lawrence Earlbaum Associates, 1987.

[3] Clocksin, W., and C. Mellish, *Programming in Prolog*, Springer-Verlag, 1981.

[4], Feyock & S. Karamouzis, LIMAP, Technical Report, Computer Science Dept., College of William & Mary, in preparation.

[5] Feyock, S., and Dalu Li, *Simulation-Based Reasoning About the Physical Propagation of Fault Effects,* Proc. of the 1990 Goddard Conference on Space Applications of AI, May, 1990.

[6] Horowitz, E., & S. Sahni, *Fundamentals of Data structures*, Computer Science Press, 1976

[7] Schutte, P., *Real-time Fault Monitoring for Aircraft Applications using Qualitative Simulation and Expert Systems,* in Proc. of the AIAA Computers in Aerospace VII Conference, Monterey, CA, October 1989.

[8] Steele, Guy, *Common LISP: the Language*, Digital Press, Bedford, MA, 1984.

# PRELIMINARY RESULTS OF INVESTIGATIONS INTO THE USE OF ARTIFICIAL NEURAL NETWORKS FOR DISCRIMINATING GAS CHROMATOGRAPH MASS SPECTRA OF REMOTE SAMPLES

**Harold A. Geller**
Institute for Computational Sciences and Informatics
Department of Physics, George Mason University
Fairfax, Virginia
and Research and Data Systems Corporation
Greenbelt, Maryland
**Eugene Norris**
Department of Computer Science, George Mason University
Fairfax, Virginia
**Archibald Warnock III**
ST Systems Corporation
Lanham, Maryland

## ABSTRACT

Neural networks trained using mass spectra data from the National Institute of Standards and Technology (NIST) are studied. The investigations also included sample data from the gas chromatograph mass spectrometer (GCMS) instrument aboard the Viking Lander, obtained from the National Space Science Data Center. We describe here the work performed to date and the preliminary results from the training and testing of neural networks. These preliminary results are presented for the purpose of determining the viability of applying artificial neural networks in discriminating mass spectra samples from remote instrumentation such as the Mars Rover Sample Return Mission and the Cassini Probe.

## INTRODUCTION

Artificial neural networks are a form of artificial intelligence that may be useful in categorizing data, particularly data that have recognizable patterns as the basis for discriminating sets within a larger group. Successful applications include optical character and speech recognition. A properly trained neural network should be able to discriminate assays using mass spectrometry in conjunction with gas chromatography. Such sample analyses are being planned for automated instrument missions to Mars and Titan. There exists a requirement to develop a light-weight, rapid capability to discriminate sample analyses and provide a first order of magnitude recommendation for further Earth-based analysis for those craft which will return sample analyses via downlinks or actual return vehicles.

The Mars Rover Sample Return vehicle will likely require its own ability to choose the samples that are returned to Earth. The Cassini probe instruments may suffer from limited transmission bandwidth thus requiring remote decisions as to what sample analyses should be transmitted back to Earth for further investigation. This preliminary investigation was undertaken to discover the feasibility of using artificial neural networks in the analyses of data and the decision making for determining the best prospects for further analysis.

Chromatography itself is the separation technique by which a sample is distributed between two phases and is resolved based upon its differential adsorption between the two phases or media (Khandpur,1981). In gas chromatography, a gas is used to transport the sample through the chromatographic column. The detector at the end of the column is used to determine the individual peaks that develop based on the time it takes for the constituents to pass through the column. Differences depend on the molecular adhesion to the column's own molecular components.

Gas chromatography is an excellent separation technique but suffers from poor identification of the constituents. Detection techniques are often used after separation in a chromatographic column to positively identify constituents of interest. The technique used to distinguish elements by their different mass to charge ratio of the ionic state is called mass spectrometry (Khandpur,1981). The sample is ionized and passed through a chamber with specific electric and magnetic fields. Detectors are placed so that peaks occur where ions are found and these peaks can tell the investigator the element which constituted the original sample (Khandpur,1981 and Message, 1984). When a mass spectrometer is used in conjunction with a gas chromatograph it is referred to as a single instrument, a gas chromatograph mass spectrometer (GCMS).

## METHODOLOGY

The Viking data is available from the National Space Science Data Center (NSSDC), located at Goddard Space Flight Center (GSFC) in Greenbelt, Maryland. The data at NSSDC, however, is currently only available from magnetic tapes which are binary-formatted for an IBM 1800.

In attempts to locate the data in a more easily read format by a neural network, such as ASCII, contact was made with numerous members of the original MIT GCMS investigation team. The GCMS data from the Viking Lander was never converted to any alternative format which led to attempts to work with the original data as provided by NSSDC.

The original six tapes sent to NSSDC were received from NSSDC by this investigation team as a single 1600 bpi tape. It contained 6 files corresponding to the original 6 tapes of data. Only the third and sixth files contains processed data from Viking Land 1 and Viking Lander 2 respectively.

There is an ongoing effort to convert this data to ASCII for the neural network, however at the same time, a limited amount of Viking Lander data was read from the microfilm archive and used for testing purposes.

For training the neural network we acquired Version 3.0 of the National Institute of Standards and Technology (NIST) PC database of electron ionization mass spectra (NIST, 1990). The search software provided with the PC mass spectra database could be used as a basis for comparison with the performance of trained neural networks. We describe two search functions which require the user to specify the peak abundances of an unknown compound. One search is by abundances of major peaks ("M" search) and the other is a search by the presence of any specified peak ("A" search).

The "M" search locates spectra in the database that display peak abundance characteristics for the largest peaks. These peaks searched are exactly the same as those of the spectrum of the unknown. This search only retrieves spectra for which the relative ordering of peak abundances that exactly match those specified.

The "A" search, adapted from a spectral search developed by Heller (NIST, 1990) requires the user to enter the mass of a peak that appears in the unknown spectra and a relative abundance range. This search allows for entering up to 10 peaks with their respective masses and abundances.

Both NIST searches terminate with a list of structures and names of compounds which are plausible matches for the unknown spectra. It is then left to the investigator to examine those spectra to determine the most likely constituents.

## TRAINING ISSUES

The back-propagation (BP) algorithm was selected as most the promising architecture for detecting the various organic molecules and their fragments. This choice was based on the need for a supervised pattern recognition algorithm that can be easily modified to accommodate different data collection scenarios. It is possible that the final architecture to be adopted for a GCMS application will benefit from having several BP modules trained independently to detect mass spectral fragments.

All pattern recognition systems require some form of training. In traditional pattern recognition systems, training consists of statistical characterization of the data using mathematical representations. For the BP algorithm, however, training consists of repeated presentations of data samples selected to be typical of the data that will be encountered in the operational setting. Each sample (or fact) is a vector of measurements from the data which is accompanied by a desired characterization. The network responds to the error signals and attempts to find a non-parametric characterization of the data set that is consistent with the training data set.

The selection of a training data set is a very important consideration. The number of samples in the training set should ideally be based upon the total number of spectra available, the *a priori* probabilities of occurrence for each spectra within Viking data, and the ability of the original data extraction techniques to present a pure spectrum.

The first criterion was bounded by the total number of spectra in the NIST data base. The second criterion could not be determined for this first analysis. The third criterion was difficult to quantify, and based solely on investigator experience. Sampling theory may ultimately provide a modicum of guidance.

The mass spectrometry technique causes the development of molecular fragments. This knowledge was used by the developers of DENDRAL, who decided to have the expert system search not for whole spectra but for fragments of spectra. It then developed a list of molecules which could form the given fragments and use additional information for narrowing the possibilities to a single or chosen few. This fragmentary analysis approach is the one often used by mass spectra analysis experts, but was too complex for this preliminary neural network implementation. The approach taken was to train the network using complete mass spectra and discover if the network could then guess the molecular make-up of the unknown.

In order to assess the performance of an artificial neural network it is necessary to apply it to one or more test sets of data having known categories (ground truth in neural network literature). If the system performance on the training and test sets is similar, the network should exhibit good generalization properties and its behavior, for example on a spacecraft on a distant planet, should produce few surprises.

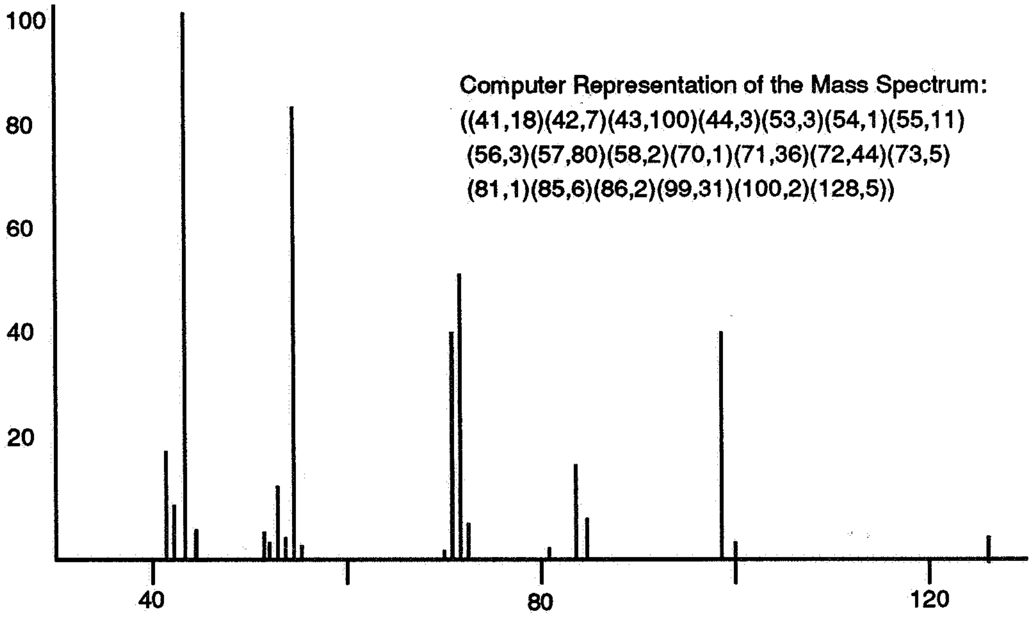
On the other hand, if the performance of the network on the test set is much worse that that of the training set, then the classifier design should be re-examined. In our study, the only case available for testing was the Viking data set. The Viking data sets themselves might have been partitioned into training and testing sets. A shortage of training data may reduce the robustness of the network whereas a shortage of testing data may reduce the confidence in the measurement of the network's performance.

### Viking Mission GCMS Datasets

On July 20th, 1976, the first Viking Lander descended upon the soil of Mars (Soffen, 1977 and Snyder, 1977). The x-ray fluorescence spectrometer, which was on board to discover the inorganic composition (Clark et al, 1977) determined the Martian soil to be composed of between 15-30 percent silicon, 12-16 percent iron, 3-8 percent calcium and 2-7 percent aluminum (Clark et al, 1977 and Toulmin et al, 1977).

The gas chromatograph mass spectrometer was said to have given an indication of a little water but no organic compounds (Biemann et al, 1977). This conflicted with results from the biology experiments which indicated the existence of microbial life (Horowitz and Hobby, 1977, Levin and Straat, 1977, Oyama and Bordahl, 1977 and Klein, 1977).

All of the data transmitted by the two Viking Landers was ultimately deposited with the NSSDC. An example of a filtered mass spectra is presented in Figure 1. A sample of a mass spectrum from the Viking Lander is shown in Figure 2 illustrating substantial amount of filtering, and is just one stage where errors can be introduced in the analyses.



Computer Representation of the Mass Spectrum:
((41,18)(42,7)(43,100)(44,3)(53,3)(54,1)(55,11)
(56,3)(57,80)(58,2)(70,1)(71,36)(72,44)(73,5)
(81,1)(85,6)(86,2)(99,31)(100,2)(128,5))

Figure 1 Mass Spectrum for 3-Octanone

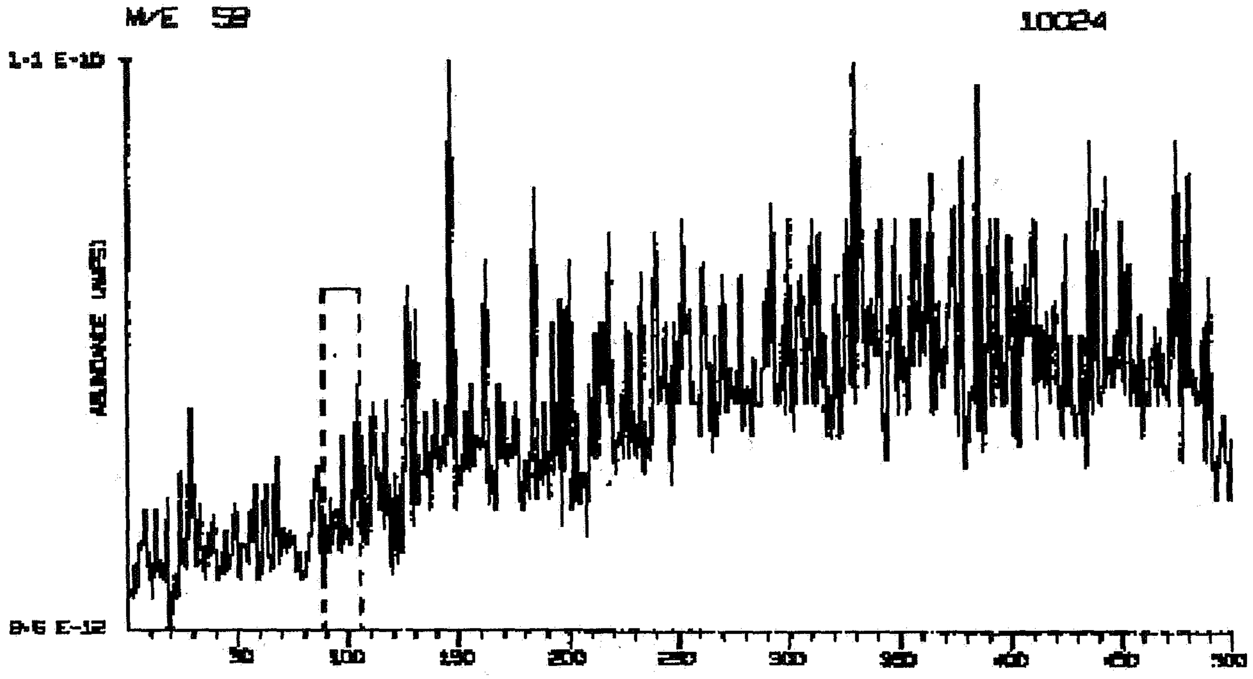


Figure 2 Viking Mass Spectra Results

## RESULTS

The neural network simulation software used in this investigation was the commercial package Brainmaker (CSS, 1990). The first attempts at training the neural network were done using a strict pattern recognition approach to the problem. The spectra are classically represented using a format whereby a list of ordered pairs represents the mass-to-charge ratio and a relative abundance figure. For example, if the highest abundance of fragmentary material from the mass spectrometer has a mass-to-charge ratio of 35, its peak would be represented by the order pair, (35,99). NIST used a scale ranging from 0 to 999 (maximum peak of 999) to represent relative peak abundances, and this was the scale adopted for this investigation.

The first attempts at training a neural network were accomplished by using a pictorial representation of the mass spectra in which individual peaks of spectra were represented by "filled" pixels at the coordinates given by the ordered pair associated with the peak. These first attempts, although yielding a trained network, were disappointing because combinations of spectra would totally baffle the network.

A different approach in the visualization on screen and the format of the data was then attempted. The on-screen visualization developed for this subsequent series used a thermometer type representation of inputs, outputs and neuron firings. Output neurons corresponded to the molecules in the training set and the higher the output neuron value, the longer the bar graph representation.

Having decided that it was most prudent to limit the number of input neurons as much as possible, even pairs of numbers were too cumbersome for data entry. The final format used included a series of numbers corresponding to the abundance at the mass-to-charge ratio represented by the node. The nodes of the first layer were ordered to make a one-to-one correspondence to the mass-to-charge ratio value.

A subset of mass spectra from the NIST database was selected as the core training set by applying two criteria:

1) The chemicals in the training set should have a history of being discovered in mass spectra from sources other than Earth
2) The molecular weight of the compound should be less than 100 (to minimize the effort required for formatting the data)

One source of GCMS chemical compounds examined in extraterrestrial samples was derived from the analysis of organogenic compounds in Apollo 11, 12, and 14 lunar samples (Flory, et al, 1972). Additional compounds were derived from an analysis of amino acid precursors in lunar samples from Apollo 14 and earlier (Fox, et al, 1972).

A list of the molecules used to train the preliminary neural networks can be seen in Figure 3. First a neural network using mass spectra of the chosen chemicals from the NIST database was trained. The network architecture used for the training was a 3-layer network with 100 input neurons, 9 hidden-layer neurons, and 14 output neurons. Associated with the 9 hidden-layer neurons is 909 weights while there are 140 weights associated with the 14 output neurons.

The input neurons corresponds to a mass-to-charge ratio (1 to 100) and each output neuron corresponds to a single chemical compound (1 to 14). The learning parameters used for the training of the network were a learning rate of 1.0, a training tolerance of 0.1, a testing tolerance of 0.4, and a smoothing factor of 0.9.

This network required 8 minutes 3 seconds to be successfully trained on a PC compatible 386SX with no math co-processor. During this training period the neural network had examined all 14 mass spectra 288 times, yielding a total reading of some 4032 spectra (14 x 288).

To determine the usefulness of adding training data of other than individual compounds, a training set was developed which also included a combination of two spectra for identification. The combination included in the training set was 50% water and 50% serine. The reason for choosing this combination was that these compounds do not have any overlapping peaks

The training of the neural network proceeded using the same network parameters and architecture. This network took 5 minutes 22 seconds and read all spectra 179 times. Each spectral set contained 15 spectra (14 individual spectra and one combination) for a total of 2685 spectra read by the network.

In comparing the weight matrices associated with these networks, we discovered that the weights for many nodes converged to the same value, although the second net converged much quicker.

Investigations into the possible corruption of data by random noise across all m/e values were performed. Figure 3 represents the interpretation of a spectrum whose m/e values (1-100) were coupled with three digit pseudo-random generated numbers (0-999).

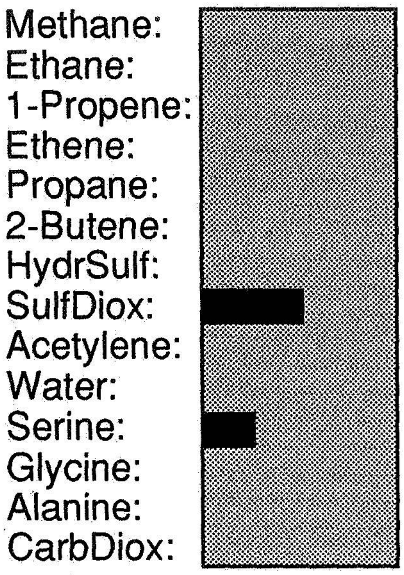


Figure 3. 3-Digit Random Values Throughout

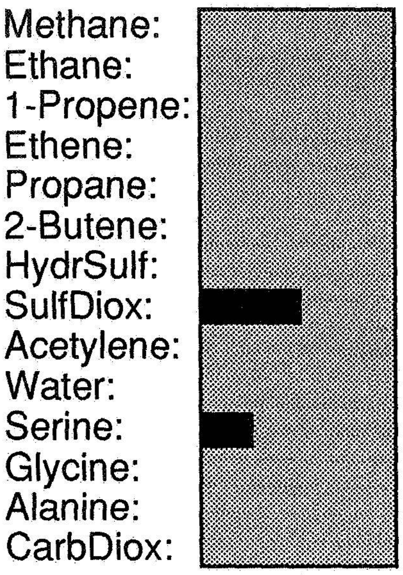


Figure 4. 3-Digit Random Values with Ethane Present

Figure 4 represents the interpretation of a spectrum which was developed by filling all m/e values with three digit pseudo-random generated numbers except at the positions of the peaks corresponding to the compound ethane.

Figure 5 represents the interpretation of a spectrum which was developed by filling all m/e values with two digit pseudo-random generated numbers except at the position of the peaks corresponding to the compound 1-propene.

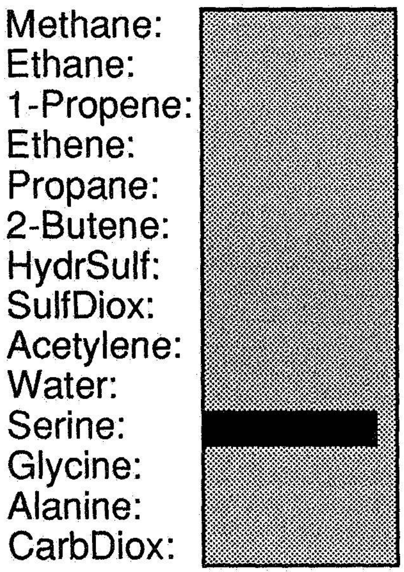


Figure 5. 2-Digit Random Values with 1-Propene Present

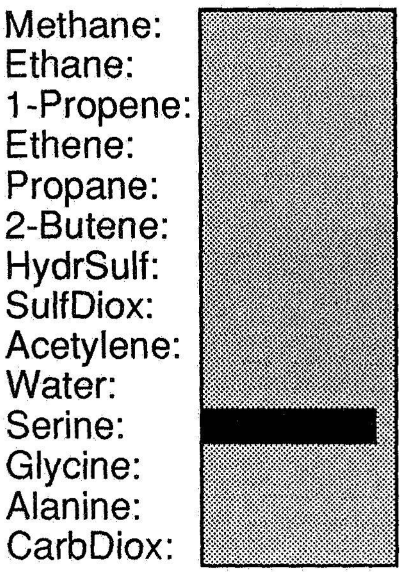


Figure 6. 1-Digit Random Values with Ethene Present

Figure 6 represents the interpretation of a spectrum which was developed by filling all m/e values with single digit pseudo-random generated numbers except at the position of the peaks corresponding to the compound ethene.

These results indicate the difficulty that the trained neural network has with the presence of random noise in the m/e peak values. This difficulty should be investigated further to determine a network's capability to identify spectra with noise similar to that which may be expected in actual remote sample analyses.

Three mass spectra from the Viking Lander were hand entered from microfilm copies of the spectra sent to NSSDC. One spectrum contained the raw data Spectra such as this represented, had to have peaks re-normalized in order to mask the peaks caused by carbon dioxide and water. When we presented the raw data to the trained network, it identified carbon dioxide (as the predominant constituent) and water (as a trace constituent) as depicted in Figure 7.

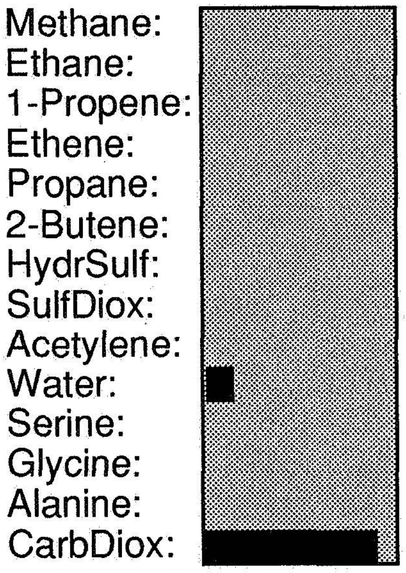


Figure 7. Viking Lander Spectrum with All m/e Peaks Present

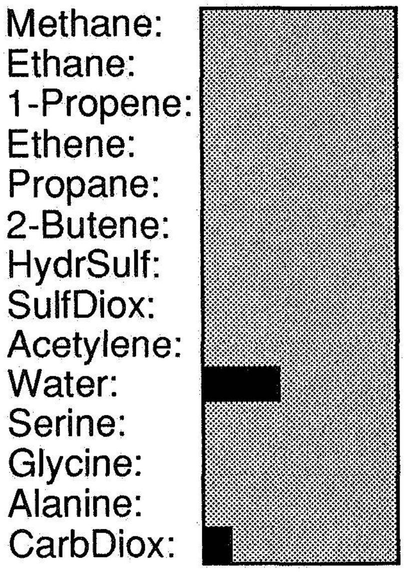


Figure 8. Tighter Tolerance Network of Same Spectrum (Figure 7)

Testing continued with a trained network set to a tighter training tolerance (0.050 vice 0.100). The second net did not do as well as the first in the identification of the constituents as can be seen in Figure 8.

Recall that the Viking data was entered by reading off values from a graph derived from a microfilm hardcopy whose error was determined to be plus or minus 1 m/e value . This error led the team to investigate the ability of the neural network when test spectra were corrupted in a similar manner.

First, the spectra for all compounds were shifted one m/e value down. That is, the first m/e ratio was dropped and substituted with the second, the second with the third and so forth until the 100th m/e value was filled with a 0 level peak value. Next, spectral peak values were shifted one m/e value upwards, in a similar manner as described.

Results from these shifted spectra were mixed. When tested with the spectra of the compounds down-shifted by one m/e value, the neural network was still able to positively identify (within testing tolerance) 8 of the 14 compounds. However, when all spectra were upshifted by one m/e value, the network identified correctly 6 of the 14 compounds.

## CONCLUSIONS AND RECOMMENDATIONS

We have sought to demonstrate the feasibility of using artificial neural networks in the discrimination of GCMS samples for the purpose of a fast and simple means for choosing interesting samples to be further analyzed in a laboratory when such analysis is not available in-situ. To this end the investigators have demonstrated the following:

1) A neural network can be trained to identify individual mass spectra of chemical compounds

2) A neural network can identify molecules whose data has been corrupted by shifts in spectral peaks

3) A neural network trained with combinations of mass spectra can accomplish its training in a shorter timeframe than one with only individual mass spectra

4) A neural network trained for identifying individual mass spectra has difficulty in interpreting mass spectra with a large amount of randomly generated data throughout the spectra

This investigation has concentrated on the second portion of the GCMS, namely the mass spectrometry. However, information on the effluent could be used in the training of the neural network which would benefit the discrimination of the sample to a finer degree than without the GC data. Separation techniques themselves are prone to certain errors as well (Silverstein and Geller, 1974). The accuracy of the Viking Lander GCMS has at least one critic, and doubt of the sensitivity of the instrument lingers to this day (Levin and Straat, 1988).

We believe that using an artificial neural network in the analysis of complex chemical data sets may yet prove to be beneficial in the future unmanned exploration of Mars, Titan and other solar system bodies. Further investigations are warranted.

**Acknowledgements:** The authors would like to acknowledge all of those that played a part in the development of this investigation and the attempts to demonstrate a viable alternative for remote analyses. Those we wish to thank include Glenn Glover (SAIC), Michael Martin (NASA JPL), Klaus Biemann (MIT), Tom Ryan (SAIC), Mary Lawler-Covell (SAIC), Hasso Niemann (NASA GSFC), Ralph Post (STX) and the National Space Science Data Center. The authors also wish to acknowledge the financial sacrifices made by Harold Geller which contributed to the success of this effort.

## REFERENCES

Biemann, K., Oro, J., Toulmin,P., Orgel, L., Nier, A., Anderson, D., Simmonds, P., Flory, D., Diaz, A., Rushneck, D., Biller, J. & Lafleur, A. (September 1977). The Search for Organic Substances and Inorganic Volatile Compounds in the Surface of Mars. *Journal of Geophysical Research* 82(28), 4641-4658.

Clark,B., Baird, A., Rose, H., Toulmin, P., Christian, R., Kelliher, W., Castro, A., Rowe, C., Keil, K. & Huss, G. (September 1977). The Viking X Ray Fluorescence Experiment: Analytical Methods and Early Results. *Journal of Geophysical Research* 82(28), 4577-4624.

CSS (December 1990) *Brainmaker User's Guide and Reference Manual 5th Edition* Grass Valley: California Scientific Software

Flory, D., Wikstrom, S., Gupta, S., Gibert, M., & Oro, J. (1972) Analysis of Organogenic Compounds in Apollo 11, 12, and 14 Samples. In Heymann, D (Ed.) *Proceedings of the Third Lunar Science Conference* (pp.2091-2108). Cambridge: MIT Press.

Fox, S., Harada, K., & Hare, P. (1972) Amino Acid Precursors in Lunar Fines from Apollo 14 and Earlier Missions. In Heymann, D (Ed.) *Proceedings of the Third Lunar Science Conference* (pp.2109-2129). Cambridge: MIT Press.

Horowitz, N., Hobby, G. and Hubbard, J. (September 1977) Viking on Mars: The Carbon Assimilation Experiments. *Journal of Geophysical Research* 82(28), 4659-4662.

Khandpur, R.S. (1981) *Handbook of Analytical Instruments.* Philadelphia, Tab Books, Inc.

Klein, H. (September 1977) The Viking Biological Investigation: General Aspects. *Journal of Geophysical Research* 82(28), 4677-4680.

Levin, G. & Straat, P. (September 1977) Recent Results From the Viking Labeled Release Experiment on Mars. *Journal of Geophysical Research* 82(28), 4663-4667.

Levin, G.V. & Straat, P. (1988) A Reappraisal of Life on Mars. In Reiber, D. (Ed.) *The NASA Mars Conference* Volume 71 (pp.187-207). San Diego: American Astronautical Society.

Message, G.M. (1984) *Practical Aspects of Gas Chromatography/Mass Spectrometry.* New York, John Wiley & Sons.

NIST (June 1990) *NIST/EPA/MSDC Mass Spectral Database PC Version 3.0 User's Guide* Gaithersburg: US Department of Commerce.

Oyama,V.I. & Bordahl,B.J. (September 1977) The Viking Gas Exchange Experiment Results From Chryse and Utopia Surface Samples. *Journal of Geophysical Research* 82(28), 4669-4676.

Silverstein,E. & Geller,H. (December 1974) Studies on the Nature of Non-Specific Staining in Nitro-Blue Tetrazolium Detection of Dehydrogenases in Polyacrylamide Gel Electrophoresis *Journal of Chromatography* 101(4), 327-337.

Snyder, C.W. (September 1977) The Missions of the Viking Orbiters. *Journal of Geophysical Research* 82(28), 3971-3983.

Soffen,G.A. (September 1977) The Viking Project. *Journal of Geophysical Research* 82(28), 3959-3970.

Toulmin, P., Baird, A., Clark, C., Keil, K., Rose, H., Christian, R.P., Evans, P.H. & Kelliher, W. Geochemical and Mineralogical Interpretation of Viking Inorganic Chemical Results. *Journal of Geophysical Research* 82(28), 4625-4634.

N91-22794

# METAMORPHOSES OF ONAV CONSOLE OPERATIONS: FROM PROTOTYPE TO REAL TIME APPLICATION

Malise Mills, Lui Wang
NASA Johnson Space Center
Houston, TX 77058

## ABSTRACT

The ONAV (Onboard Navigation) Expert System is being developed as a real time console assistant to the ONAV flight controller for use in the Mission Control Center at the Johnson Space Center. Currently the entry and rendezvous systems are in verification, and the ascent is being prototyped. To arrive at this stage, from a prototype to real world application, the ONAV project has had to deal with not only AI issues but operating environment issues. The AI issues included the maturity of AI languages and the debugging tools, what is verification, and availability, stability and size of expert pool. The environmental issues included real time data acquisition, hardware suitability, and how to achieve acceptance by users and management.

## INTRODUCTION AND PROBLEM STATEMENT

What follows is a description of the development of the ONAV expert system. This project could aptly be titled the Agony and the Ecstasy. The ecstasy being the privilege of working with a group of highly talented and dedicated people, who persevered through some very trying times. Of working on the state of the art project that would enhance the quality and reduce costs of mission support. The agony portion of the project consisted of attempting to develop an real-time operational system when the state of the art of the supporting technology was still in its infancy and when; for the most part, there existed institutional pressures not conducive to change and innovation. This paper will trace out the development of the ONAV system and hopefully highlight some of the pitfalls to avoid in development of such a system.

The idea of the ONAV Expert system was conceived in the summer of 1986 by the Mission Support Directorate (MSD); now Information Systems Directorate (ISD), at the Johnson Space Center. MSD's goal for this project was to develop and implement a real-time expert system in the Mission Control Center. The ONAV position was selected for this purpose because it provided a well defined and relatively small problem domain. Also several of the software engineers at MSD had previous ONAV experience. However, to develop this project MSD needed to elicit the support of the ONAV flight controllers, who work in the Mission Operations Directorate (MOD) . At the beginning there was some opposition from MOD since we had flown 25 flights previously without such a system, therefore we could and can fly future missions without such support. But as with many projects, it was through the perseverance of several people that the project stayed on course.

## ONAV FUNCTION

The Onboard Navigator (ONAV) is a back room position that supports the Guidance and Procedures Officer (GPO) in the Mission Control Center. The ONAV's principal responsibilities are to monitor the health of the onboard state vector and navigational aids that are used to update the state vector and make recommendations to the GPO for actions to maintain the state vector. The state vector represents the orbiter's position and velocity at a given time and is used in various guidance and control functions on the orbiter. The state vector is propagated using the inertial measuring units (IMU's) and is updated by a variety of navigation sensors through a kalman filtering scheme. The sensors used are dependent on the phase of flight. ONAV supports three phases of flight; ascent, entry and rendezvous. During ascent only the IMU's are used; during entry the IMU's along with the tactical air command and navigation system (TACAN), air data transducer assembly (ADTA), a drag altitude model and the microwave scanning beam landing system (MSBLS) are used. Rendezvous uses the IMU's, star trackers and the rendezvous radar. Basically, the ONAV job entails monitoring several digital displays that provide information on the status of the various navigation sensors and the onboard state vector. Approximately 180 parameters are normally monitored. The ONAV, based on this information, makes recommendations ranging from incorporating the sensor data into the onboard filter to recommending a ground derived state vector be uplinked to the onboard system. To be able to perform this function an ONAV must have an understanding of how the onboard software operates, how the sensors function, crew procedures and knowledge of kalman filtering. In addition the ONAV must be able to rapidly determine what information on the displays is important, which varies as a function of time, what failures have occurred, and what sensors are available.

## OBJECTIVES AND GOALS OF SYSTEM

The objective of the system is two fold; one is to use the system as a console assistant and the other is as a training tool. Currently each phase of the mission requires two ONAV's for support; a lead and an ONAV2. The lead's responsibilities are to coordinate all information from the console, communicate with the GPO and make all pertinent calls. ONAV2 logs information and provides backup to the lead. As the system matures and both the GPO's and ONAV's become more confident with it, the goal is to eliminate the ONAV2 position. Also the system should enhance the quality of mission support. This is being accomplished in a variety of ways. The first method is the automatic logging capability of events and recommendations with both an altitude and a time tag that the system generates. Previously the ONAV's would log these events by hand and if several critical problems occurred at once it became difficult to keep up. The system will eliminate that problem. Another enhancement is the incorporation of color graphics in the user interface. This provides an easier to read screen and significantly aids in problem recognition and resolution by providing trending analysis.

Probably the most critical enhancement provided by the system lies in its ability to monitor all of the sensors at all times. At certain periods, one sensor may be more critical than others. As a result, the ONAV's attention tends to focus on the status and the functions of that one sensor and loses track of the others. The expert system, however, has the ability to continuously monitor all of the available sensors while still providing increased focus on the specific sensor that is providing critical data. Over 600 parameters are monitored by the entry system and 425 parameters for the rendezvous system. We expect the ascent system will be required to monitor approximately 400 parameters. This function has already proven to be highly valuable during flight simulations conducted by MOD, when the system alerted controllers to a secondary problem.

The second objective for the system is to use it as a training tool. Currently, training an ONAV2 takes over a year. An additional nine months is then required for ONAV1 certification. A major impact to the length of time that the training process takes is the number of simulation hours available. A new trainee can expect, at most, four hours a week in simulation time. The expert system will lessen the trainee's dependance on simulations by providing the capability of running training cases in an office environment. The biggest advantage of the use of the expert system as a training tool, however, lies in the establishment of the rulebase and its knowledge capture feature. When running as a training tool, the expert system will be equipped with a knowledge browser and rulebase rationale. This will allow the trainee to assimilate the information at his own pace and as a result he will not be as dependent on having an experienced person available to answer questions. Finally, by establishing a single set of agreed to rules the certified rulebase will provide a source of consistent training.

## IMPLEMENTATION

The development of the rules for the ONAV expert system was a joint effort between the domain experts of MOD and knowledge engineers from MSD. The ONAV domain knowledge was captured through a series of meetings where the ONAV experts presented information to the knowledge engineers in a classic classroom setting. Initial meetings focused on the system functional overview. The results of the meetings were converted to different design strategies and prototypes, which could be critiqued by the experts. Prototype development then, served as the feedback mechanism for the meetings. During the development phase, when the knowledge engineer understood the basic operations, they would generally develop the agenda for the meetings on a particular topic and the experts provided the detail knowledge. The meetings were scheduled once a week two to three hours for the first three months, and after were held less frequently. By the end of six months we had the first prototype with a minimum user interface. But the overall system design structure had been selected and tested.

It turned out that the ONAV normal task functions were fairly straight forward; however, detecting anomalies in the subsystems and component failures recommending proper response required detailed expert knowledge. The overall system design was completed in the first year of the development phase. This structure resulted from the basic nature of the ONAV task and from the modularity guidelines of any good system engineering approach. Four functional components of the expert system can be identified: (1) Fact assertion, (2) monitoring, (3) analysis, and (4) output. In addition, there is a fifth non-expert system component that is a part of the overall ONAV system called "data preparation".

The data preparation (data prep) component receives information from the operational environment and performs three functions:

- Collects the information required by the expert system
- Performs any computations required on the data
- Filters and transforms that data into a form suitable for the expert system.

In short, the data prep component converts the numeric instantaneous data points to a set of symbols which reflect the environment. The fact assertion component simply takes the prepared data and puts that data required by the expert system into the fact base. The monitoring component generates intermediate conclusions and statuses of the individual subsystems that ONAV observed and manages. This component simply detects environment state changes and pass them on to the next component. The analysis component performs an overall assessment of the current status, (i.e., makes use of the data from the monitoring phase) confirms the subsystems status, and/or recommend actions to prevent further degradation of the state vector. The analysis phase is the heart of the knowledge base, because this component captured the ONAV expertise. Specific knowledge implementation such as letting TACAN take out the state error rather than doing an uplink to the orbiter, or deselecting the line replaceable unit (LRU) when the onboard data bus has a commfault, are programmatically captured, and kept. The output component controls the sending of the notices and/or recommendations to the ONAV expert system console.

As for the look and feel of the system, our philosophy was to keep it as simple and clean as possible. To achieve this we employed the pop and shoot method. This method consisted of a series of pop-up menus or toggle switches activated by the mouse. This provides a format that is easy to read and allows the operators to keep their eyes on the screen, a required feature in the highly dynamic environment of orbiter high speed operations. Messages appear in windows that can contain up to 20 messages and are scrollable. Status and quality lights take full advantage of color and also contain text to insure that the operator understands the meaning.

As a real time console assistant, the expert system requires real time telemetry and trajectory data from the Mission Operations Computer (MOC), using a data stream called Generalized Data Retrieval (GDR). The GDR data is actually retrieved by a program called GDR_driver, which reads the data from the MOC and writes it into a shared memory segment. This shared memory interface is based on a model developed at JSC for use by real time applications (ref. Workstation Application Interface to Data Source Interface Agreement). This model is generic and modular and it provides a standard mechanism to access different data sources.

## CERTIFICATION

The certification/validation scheme we devised for the system was established at three different levels. The first level consists of the component level; data acquisition (GDR), data prep, rulebase, plots and the interface. Its intent is to insure that the program is functionally correct, ie. that data is correctly gathered, that the rules fire if presented with appropriate actions and events, and that results are correctly displayed to the screen. The second level consists of integrated system testing and overall certification. The intent of this level of testing is to insure that the various constituent parts of the system function in an integrated manner and, more importantly, that the recommendations made by the system accurately model the expert's mental concepts of the state of the system and are the correct responses to environmental changes. The third level consists of an integrated workstation certification. This last level is accomplished because of the critical nature of the workstations and is to insure that this application will not interfere with any other workstation or mainframe applications.

For the GDR certification, we collected data during a sim at several time points. We took hardcopies of the data from the mission operations computer (MOC). For each time point we then compared the data from the MOC to the data we received at the workstation. Once we were certain that we were getting good data from GDR, we then went through the same process with data prep. The next step in component testing has been the verification and validation of the rulebase, which has been the most difficult of the tasks.

To attack the problem of verification, we first had to start by deciding what verified meant. The existing literature, at the time, on verification and validation we found to be lacking for a rulebase of this size, approximately 300 rules for entry and 500 rules for rendezvous. The final decision, after much debate, was the construction of an error matrix that contains 48 errors that a certified ONAV would have to be able to deal with to be certified. For the rulebase to be declared verified, the system must act correctly at least twice for each of the errors.

The method that we have been using to verify the rulebase has largely been dictated by the source of test data. The optimum method would have been to run a nominal case and then introduce a specific error in each of the following runs. Due to the complexities of the problem environment, the only source of data for the system is simulations or missions. However, the goal of simulations is to train flight controllers, therefore we are at the mercy of the training area as to what errors occur in a sim. Consequently the verification is somewhat of a hit or miss operation. We log and evaluate testcases when they are available, not necessarily in a controlled, selected fashion.

When a sim run is selected, it is first delogged. In this manner actual copies of the controllers screens are obtained from the run. Next, an expert will go through the delog and determine what occurred in the case and what calls and events the ONAV flight controller would have made and noted. The next step is for the expert to run the testcase through the expert system. At this time, any errors or discrepancies are noted. The expert then writes a log with the errors and discrepancies and what the correct action should have been. This is then passed to the software engineers, who in conjunction with the experts, work on correcting the problems. After the software engineers are done, then the corrected rulebase is passed back to the experts. The testcase is then replayed, and if all errors have been corrected and no new ones appear then the new rulebase is used in replaying all previous testcases. If everything checks out then the that testcase is signed off by all parties.

In the event any questions arise that concern procedures or requirements, the action is referred to the ONAV working group. The ONAV working group was formed, with representatives from all the functional ONAV and software engineering areas, to focus the collective knowledge of the ONAV community and provide an authoritative method of resolving requirement conflicts or procedural debates. In this fashion controllers were provided input into the generation of the rulebase and a feeling that they know and understand what corrections have been made and the methodology that is incorporated in the rulebase. In addition the working group minutes serve to document the decisions made and provide the knowledge engineers a paper trail that they can refer to when creating/modifying the rulebase.

Once all the various components have been verified, the system will be ready for full certification. At this point the system will be treated just as any flight controller for certification. To obtain final certification, the system will be run through a full set of entry sims with an GPO and ONAV1 monitoring. The GPO and ONAV1 will then evaluate and determine final certification. Also during this phase the system will be checked to verify that it will not interfere with any other workstation application.

Since not only expert systems but also workstations represent a new way of console operations, the system will be implemented in three phases. The first phase, which is going on during rulebase verification, is for a third controller to monitor the system behind the current operator. This allows us to run the system during actual sims and flights without conflicting with console operations. Once the rulebase is verified, the system will then be moved on console and be monitored by the ONAV2. This phase will allow us to do the final interface work and answer questions such as, is the system easy to read and does it convey all the information necessary for console operations. Once all operators become comfortable with the system then at that point it will be ready to replace the ONAV2.

## PROBLEMS

From our experience on the project, time is the most critical element on the project. Time provides the luxury of careful scrutiny of the system but if time is not carefully allocated it can rob the project of its momentum. The following problems represent areas that forced the project off its path and consequently slowed the development of the project.

As the name expert system implies, an expert is an integral part of developing an expert system. When this project first began, there were two ascent/entry experts and no certified rendezvous experts. For the projected flight rate at the time, six ascent/entry controllers were needed and two rendezvous controllers. Due to the low numbers, the priorities of the available experts were flights, training, analysis and finally expert system development. So not only were there a limited amount of experts, their availability was extremely limited also. At one point the only expert working on the entry system quit, leaving roughly a year and a half where no expert was available. These factors stretched the development out over a considerable length of time. This led to problems in the software engineering side, in that the programmers became restless with the lack of progress on the system. It consequently contributed in the decision by several programmers to leave the project. The expert system, in fact, had fallen prey to the very thing the system was trying to protect against, that of a lack of a knowledge base. Due to this problem, the luxury of time for a quick development of a prototype, at the start, was denied the project, consequently robbing the project of its needed momentum.

Due to the drawn out development time and turnover problems several different groups of experts worked on the entry system over time. This meant that the rulebase represented an inconsistent set of methodology and at times without documentation a lack of understanding of why things were done the where they were. It was in response to this, that the ONAV working group was formed. Its function is to try to reach a consensus among current operators and document that methodology. The working group concept brought a sense of stability into the project because it as a organization remained constant and left a documented trail.

Another obstacle faced by the project was the uncertainty over the platform that would be used for the final product. One of the goals of the project is to be used as a console assistant. The system is to reside on a Mission Control Center Upgrade (MCCU) workstation. Unfortunately, our project and the MCCU project have been developed in parallel. During the evolution of MCCU the final platform went from MASSCOMP to SUN back to MASSCOMP, from a two MIP's to an eight MIP's machine and from a non-color to a color graphics terminals. Through it all we tried to keep the system as portable as possible and tried to get our requirements into the MCCU project. As a result of our philosophy of portability, the project did end up with some throw away code. The interface was done in four different versions; Curses, MC-Windows, SunView, and X-Windows versions. Even at this time, the MCCU's lack of robustness has caused the expert system setbacks in gaining controller confident since 90% of our sim failures can be attributed to workstation problems. It is difficult to avoid this type of problem when working in a high tech environment, on what might be called the bleeding edge of technology. The best that we can do is be alert to changes and try to influence any changes to our environment. We believe the fact that the systems were worked in parallel helped since our system was ready to move on to the configured workstation. Consequently we were able to evaluate workstation performance. The time lost working workstation problems, though, stole valuable time from user performance time.

As for any system data is the life blood of the system. There existed two data problems for the ONAV expert system, availability and access. As previously stated the only acceptable data source, due to the size of the problem environment, is from simulations and flights. This proved to be a constraint on our development from the start. The main problem, however, was in getting access to the data required. The system requires telemetry data from the orbiter, for onboard systems and state vector evaluations, and data from the MOC for output of high speed ground filter data. Several source existed from which telemetry data could be obtained. GDR, however, was the only source for the ground filter. A longer delay in achieving access to GDR data than anticipated occurred due to MOC integrity concerns. This caused an even larger delay between prototype development and rulebase verification. Possibly if we had tried to get a better understanding of the system's data requirements closer to the start of development; we may have been able to piece together a set of nominal testcases from some analysis tools. This would have given us a leg up on verification but not solved the whole problem. Also from our experience with verification the system's data requirements can expand through this process. This one factor stole the most momentum from the project and was one of the most frustrating experiences to endure. The system was all dressed up with no place to go.

The C-language Integrated Production system (CLIPS) expert system shell was used in the development of the expert system. The ONAV project received one of first beta copy of CLIPS (release 3.0) for the prototype of the ONAV project. Even in the beta software, CLIPS inference engine was very robust. However, as the expert system development continued, more rules were added to the system. The rule based interactions became more complex. CLIPS at this time did not provide adequate debugging methods to deal with large knowledge bases, but in fact the state of the art for expert systems in general was lacking in this area. This was a particular problem because the expert system has to be embedded with other modules, such as the real time data acquisition, and user interface modules which added another layer of complexity. For a long time, tracking down a typographical error was very difficult. Since then, CLIPS development team has developed additional tools such as the cross-reference, style, and verification (CRSV) tool, CLIPS window interfaces environments and also added additional syntax to the pattern language to deal with some of the problems. Finally, much research is still focusing on finding the appropriate methodologies to perform verification and validation on expert systems. Certainly, CLIPS will also evolve along with the technology. Those are some of the costs that one has to realize and eventually justify for its value of the project.

One area of particular concern was the acceptance of the system by users and management. One criticism from management was that the system would erode controllers skills, that a reliance on the system would eventually cause destruction of the ONAV knowledge base. However, our response was if the certification process for a controller is adequate then there should not be any erosion in skills. Users were also reluctant to use the system because the system did represent change and would require new training. Our attempt was to bring in the users at the beginning, keep them involved and place the interface design in their control. Also all rulebase changes had to be blessed by the expert. In these ways the user community became totally involved in development and responsibility for the project thus increasing acceptance by the users. The major factor in acceptance, however, is simply time. Time for the flight controllers to use the system and watch as the system's performance proves itself worthy of confidence. Time for management to accept that the system will not be death of flight controllers as we know them. Time for the system to become fully integrated into console operations.

## CONCLUSIONS

To arrive at this point in time, where users are beginning to use the system, where it is actually being of benefit to operators, has been, as chronicled in the preceding passages, somewhat a tortuous path. In hindsight, a critical element in the development of the project was the length of time between rulebase development and data flow. The length of time for our project proved to be much longer then the ideal, causing us severe problems but, ultimately, not disastrous ones. What we should have done was have the assurance of the availability of experts and a first cut of data before starting rulebase development. Without this, a rapid development of a realistic prototype proved impossible, which prevented the users from actually using the system. Time on the system and with the system, by the users, as we have experienced is crucial for the development of the system. The working group concept should also have been used from the start of the project to facilitate documentation and provide stability for the system.

Time, it turned out was an enemy of this project. Too much was spent waiting; waiting for experts, waiting for workstations and waiting for data. As previously concluded, prototyping quickly, and keeping everyone involved is critical. Luckily, however, time will come to our rescue. We have spent a large amount of time on certification. This time has only increased the confidence of the controllers. And the more time spent in console operations, the greater the acceptance. Hopefully the opportunity will present itself that some of these lessons can be applied to other systems and in time, we can learn from the past and not just ignore what we have learned.

References:

[1] Giarratano, J.,*The CLIPS User's Guide.* NASA Document, June 1988

[2] Riley, G., *CLIPS Architecture Manual.* NASA Document, JSC-23047, June, 1988

[3] *Guidelines and System Requirement for the Onboard Navigation Console Expert /Trainer System.* NASA Document, JSC-22433, June 1986

[4] Liebowitz, Jay, *Expert Systems with Applications Volume 1* November 1990

[5] AAAI-90 Workshop on *Knowledge Based System Verification, Validation and Testing* Boston, Massachusetts July, 1990

[6] *Knowledge Requirements for the Onboard Navigation (ONAV) Console Expert/Trainer System: Entry Phase.* NASA Document, JSC-22657, September, 1988

N91-22795

# LESSONS LEARNED IN THE DEVELOPMENT OF THE STOL INTELLIGENT TUTORING SYSTEM

Thomas Seamster, Ph.D
Mr. Clifford Baker
Carlow Associates Incorporated

Mr. Troy Ames
NASA Goddard Space Flight Center

## ABSTRACT

This paper presents lessons learned during the development of NASA's Systems Test and Operations Language (STOL) Intelligent Tutoring System (ITS), being developed at NASA Goddard Space Flight Center with support by Carlow Associates Incorporated and Computer Sciences Corporation. The purpose of the intelligent tutor is to train STOL users by adapting tutoring based on inferred student strengths and weaknesses. This system has been under development for over one year and numerous lessons learned have emerged. These observations are presented in three sections, as follows. The first section addresses the methodology employed in the development of the STOL ITS and briefly presents the ITS architecture. The second presents lessons learned, in the areas of:

- intelligent tutor development
- documentation and reporting
- cost and schedule control
- tools and shells effectiveness

The third section presents recommendations which may be considered by other ITS developers, addressing: access, use and selection of subject matter experts; steps involved in ITS development; use of ITS interface design prototypes as part of knowledge engineering, and; tools and shells effectiveness.

## BACKGROUND

This paper presents lessons learned during development of the STOL ITS, which is being developed to assist NASA control center personnel in learning STOL.

The STOL ITS is currently being designed to provide the Gamma Ray Observatory (GRO) Flight Operations Team (FOT) with introductory and refresher training/tutoring on STOL and its main applications. The initial orientation of this effort was to provide intelligent tutoring in the context of complex dynamic systems. This initial effort was concerned with the application of ITS technology to improve system performance and safety in supervisory control. The emphasis was on the modeling of operator's intentions in the form of goals, plans, tasks, and actions. More recently, this research has expanded to include a review of ITS technology as it may be applied to the tutoring of command and control languages (Eike, Seamster, & Truszkowski, 1989).

A related effort is taking place at NASA Johnson Space Center (JSC) through the development of PD/ICAT, an ITS to train Mission Control Center Flight Dynamics Offices to deploy a specific type of satellite for the Space Shuttle (Loftin, Wang, Baffes, & Hua, 1988). PD/ICAT is part of a larger effort to develop a general architecture for intelligent tutoring/training systems to train controllers in the performance of mission-critical tasks. This architecture consists of a blackboard that serves as a communications interface for the following system's components: user interface, expert module, trainee model, training scenario generator, and a training session manager. The developing STOL ITS design utilizes the NASA/JSC general architecture with several modifications that will allow for expanded capability and additional flexibility (Seamster, 1990).

## Purpose

The purpose of the STOL ITS is to facilitate understanding, development and application of STOL directives and procedures. Since this project is primarily a research effort (as opposed to a development effort), several research questions are being addressed, including:

1. the feasibility of developing a general purpose STOL tutor which can be effectively utilized in a wide range of POCC environments;
2. evaluation of the relative effectiveness of alternative knowledge representat-ions for tutoring a command and control language;
3. evaluation of the relative effectiveness of alternative user interface modes for students with varying skill levels.

A major element of this project is to demonstrate the effectiveness of ITS technology in a NASA control room environment by selecting a set of representative STOL tasks which have application across a wide range of POCCs.

## Project Phases

STOL ITS development is proceeding in three main phases, as follows.

Feasibility Stage - The focus of this stage involves evaluating the feasibility of developing a general command language ITS. Two aspects to feasibility have been addressed: technical and cost. Technical feasibility refers to reasonableness of developing an ITS to tutor STOL. In this context, feasibility is determined by the availability of Domain Experts (DE's) capable of expressing the nature of their expertise in a form which can be translated into an expert system.

Cost feasibility refers to the probability that the costs incurred in developing the ITS can be recovered through improvements in POCC STOL training. In order to justify the development costs associated with an ITS, there must be a relatively high number of students processed through the system. Since no individual POCC is likely to have a sufficiently large number of students to justify the cost, the ITS must be applicable across several operating environments. In order to meet this objective, the operational version of the STOL ITS is being designed to be portable and modular, emphasizing a decomposition where specific command language functions are separated from the general command language tutoring functions.

Research/Extensibility Stage - The object-ive of this stage is to expand the rule set to encompass a relatively complete set of DE cognitive functions. One of the main objectives of this stage has been to develop the various ITS modules to the point that they can interact (i.e., exchange data) to perform the functions of an intelligent tutoring system. Consistent with the objective to reduce costs, these modules operate under a common operating system, and, where possible, are developed with existing NASA data, rule sets and development tools. The general ITS architecture developed at NASA/JSC (Loftin et al., 1988) has been used. A parallel effort has been to develop and validate the user interface for the ITS. This effort consists of an iterative series of develop-ment / demonstration / revision cycles, in which potential users of the ITS evaluate and comment on the user interface. The focus of this effort has been directed at investigating the relative efficacy of alternative forms of problem/knowledge representation for students with varying levels of experience and expertise.

Field/Demonstration Stage - The objective of this phase is to demonstrate/evaluate the viability of the STOL ITS. As part of field demonstration of the GRO STOL ITS, the STOL ITS provides the trainee with several aids to facilitate learning. The first aid is an alternative representation of STOL which is graphically oriented, and which consists of dynamic icons and graphics representing the various elements of the GRO environment. This representation is used to facilitate learning STOL and is treated as an intermediate step between the trainee's current way of thinking about the system and the way he or she will use STOL in the control room. Another aid is a hypertext glossary of STOL directives, containing semantics, syntax and examples. Finally, a STOL Certification tool has been developed which measures a students STOL/GRO proficiency and provides practice in use of the STOL language as applied to GRO.

## Knowledge Acquisition Process

The STOL ITS is designed primarily for the mission analysts of the GRO FOT. There are to be 9 individuals serving in this capacity. In addition, the STOL ITS, in its initial version, may be used by spacecraft engineers and guest scientists. With some modification, the ITS may also be used by Multi-Satellite Operations Control Center (MSOCC) operators, and by analysts on other missions. As was mentioned in the introduction, STOL ITS is being developed to ultimately serve a number of different missions, and a number of its modules may be used as the basis for tutors of other NASA command and control languages.

The GRO FOT mission analysts include the Operations Controllers (OCs) and the Command Operators (COs). The OC is the senior person who directs the activities in the GRO Missions Operation Room (MOR), and the CO executes the specific STOL commands. The OC is the senior person who directs the activities in the MOR, and the CO executes the specific STOL directives. In addition to these two operators, there is a mission planner present during the day shift and a group of engineers who will be in the GRO MOR when required. This paper concentrates on the tasks of the two primary operators, the OC and CO. The OC evaluates the status of GRO related systems and subsystems. The OC uses at least three CRTs to monitor the different types of status data. The OC also interacts with a number of people in the execution of his job. In addition to the CO, the OC interacts with other network operators via a voice link communication, making the communication system an important part of the OC job. Given this background, the ten steps of the STOL ITS Knowledge Acquisition Process are presented in Table 1.

**Table 1. Ten Steps in the STOL ITS Knowledge Acquisition Process**

1. Extract STOL and MSOCC terms from documentation
2. Sort and cluster analysis of STOL elements
3. Identify key directives and SMEs for the project
4. Identify the key concepts and potential tutor problems
5. Rate and select key problem/solutions for the demonstration tutor
6. Develop problem representations
7. Gather SME protocols for set of problems and transcribing
8. Analyze first set of protocols and developing pseudo-code rules
9. Expand and refine first set of pseudo-code rules and provide novice misconceptions
10. Translate rules into CLIPS

A collateral step to the knowledge engineering (KE) process was the development of the STOL ITS user interface. This interface, developed in HyperCard, controls the presentation of information to the STOL ITS students, and interfaces with the CLIPS module which house the ES portions of the ITS.

## STOL ITS Tools and Interfaces

STOL Certification Tool. The purpose of this tool was threefold. First, it was developed to refine and verify STOL user error classifications, such as error tendencies of students as a function of STOL experience and spaced based platform experience. Secondly, it was developed to measure the progress of students as they used the STOL ITS. Finally, it has been used as method to extensively practice issuance of STOL commands to GRO, with feedback provided to students as to the efficiency of their command operations. The Certification tool:

- collects error data on STOL users (commands, arguments, syntax)
- provides a basis for developing and validating STOL ITS student models
- provides a tool for certification of STOL users after and during training (including STOL ITS training)
- practices student issuance of STOL commands to GRO

The certification tool has a simple interface and architecture. Four sections are provided for student testing/practicing of STOL commands for GRO major mission sections. These are: Introductory; Prepass; On Pass, and; Post Pass.

Within these sections, there are approximately 200 questions which can be posed to students, with one to three answers being considered correct for each question. In the course of a session, the Certification toll poses a problem, and the student responds, after which the tool determines whether the response is correct, logs the data, and poses a subsequent problem. Figure 1 shows a certification tool screen with a student response.

Hypertext STOL Glossary. The student may also access a detailed Hypertext STOL glossary, which presents:

- STOL directives
- Aliases
- Required and optional arguments for directives
- Descriptive information related to directives
- Modal use of directives

The glossary is used in the course of a session when a student has difficulty in responding to a posed question. Figure 2 presents the basic interface for the STOL Glossary.

STOL ITS Interface. Development of the STOL ITS has placed considerable emphasis the design of the interface. This has been for the following reasons:

- ITS development matured to the point of becoming user-centered
- complex, relational information to be presented
- the use of the interface to gather student data
- the use of the interface to evaluate different tutoring strategies
- the use of the interface in knowledge acquisition
- The interface provides a point of convergence for the various STOL ITS modules

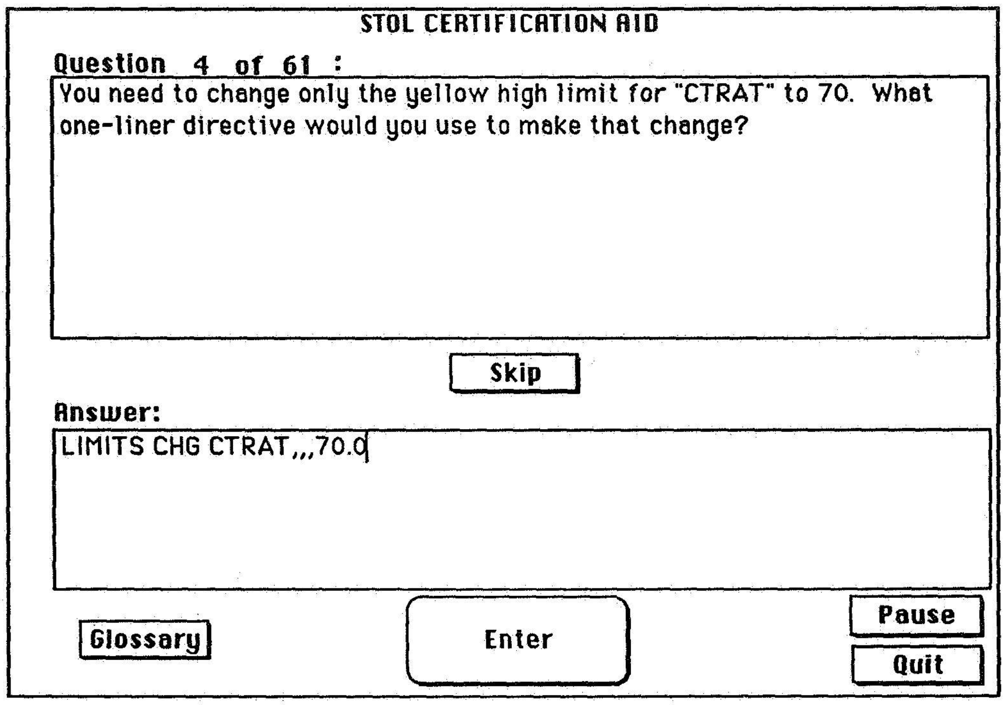
Figures 3 and 4 present example of STOL ITS interface screens.

## LESSONS LEARNED

Lessons learned in the development of this ITS are related to: intelligent tutor development in general; documentation and reporting; Cost and schedule control, and; Tools and shells effectiveness.

### General Lessons

Subject Matter Expert Access. It was estimated that the development of the STOL ITS would involve a number of interactions with the user community. This community, the GRO FOT, consists of 20 individuals who would serve as Subject Matter Experts (SMEs) and novice users. The SMEs were to serve as the main source of STOL information during the knowledge acquisition phase. Novices would be used to evaluate the prototype user interface as well as the prototype STOL ITS. A total of 200 hours was requested from the GRO FOT. The GRO FOT's initial reaction was that it would be difficult to provide that many hours. Part of the difficulty was that the GRO FOT would be in the middle of End-to-End (ETE) tests in preparation for launch. It became evident that if the GRO FOT had to provide 200 hours, it would not be able to participate in the STOL ITS development process. Because of this, the SME and novice trainee needs were reevaluated, and a



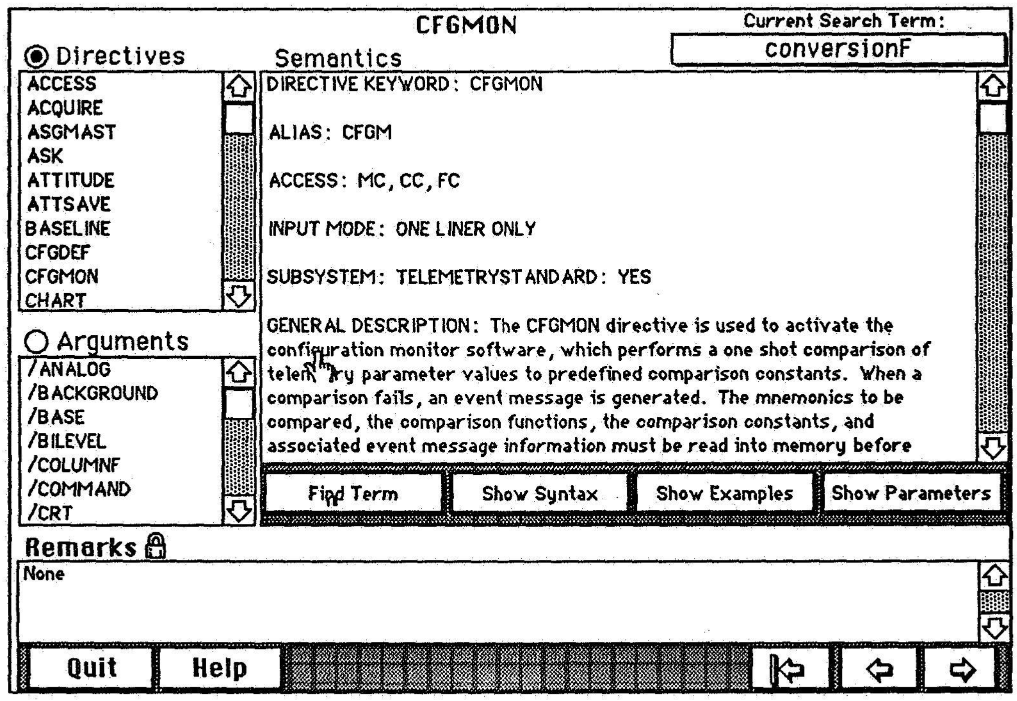
Question 4 of 61 :

You need to change only the yellow high limit for "CTRAT" to 70. What one-liner directive would you use to make that change?

Answer:

LIMITS CHG CTRAT,,,70.0

**Figure 1.**
**Certification Tool Interface with Student Response**



Current Search Term: conversionF

● Directives

ACCESS
ACQUIRE
ASGMAST
ASK
ATTITUDE
ATTSAVE
BASELINE
CFGDEF
CFGMON
CHART

○ Arguments

/ANALOG
/BACKGROUND
/BASE
/BILEVEL
/COLUMNF
/COMMAND
/CRT

Semantics

DIRECTIVE KEYWORD: CFGMON

ALIAS: CFGM

ACCESS: MC, CC, FC

INPUT MODE: ONE LINER ONLY

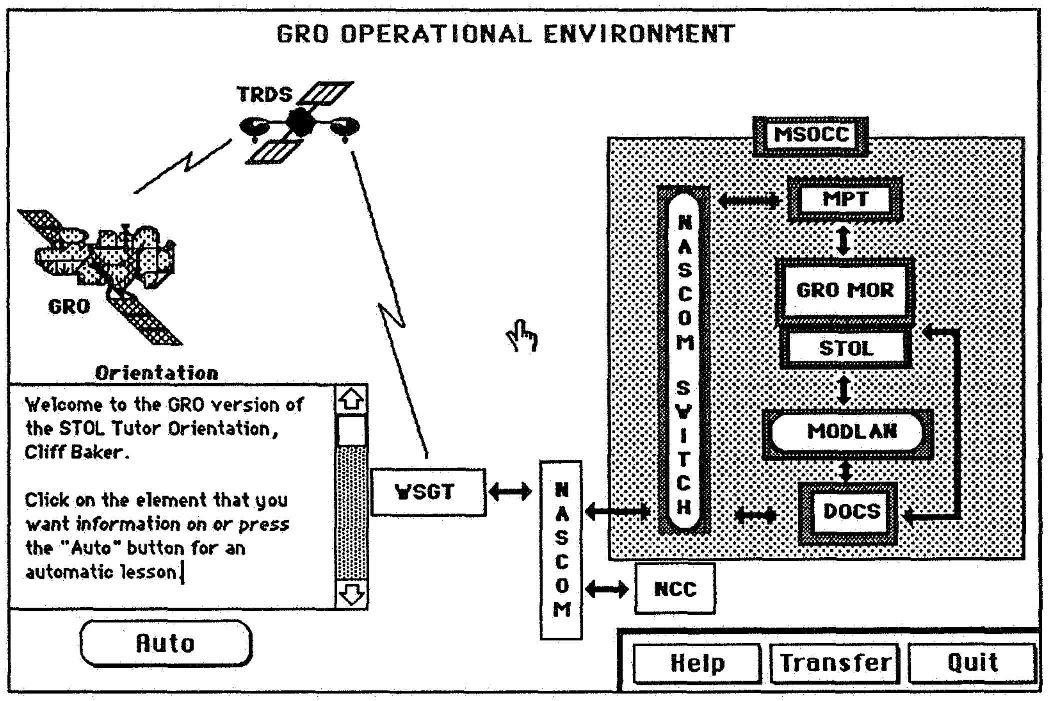
SUBSYSTEM: TELEMETRYSTANDARD: YES

GENERAL DESCRIPTION: The CFGMON directive is used to activate the configuration monitor software, which performs a one shot comparison of telemetry parameter values to predefined comparison constants. When a comparison fails, an event message is generated. The mnemonics to be compared, the comparison functions, the comparison constants, and associated event message information must be read into memory before

Remarks

None

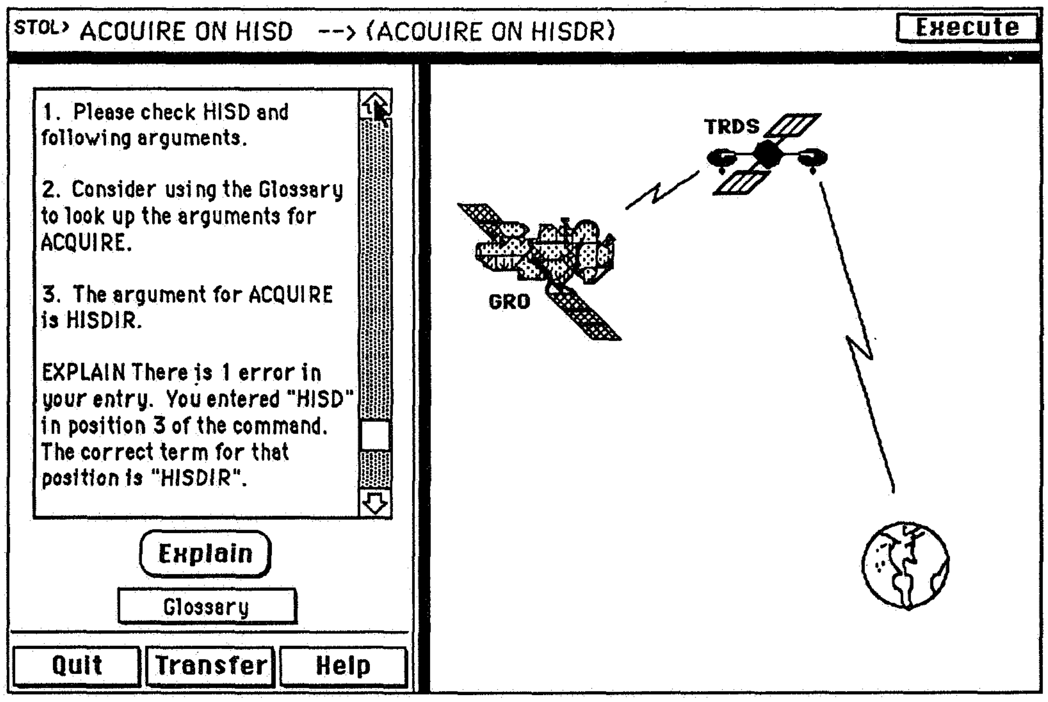
**Figure 2**
**Basic Interface of the STOL Glossary**



Welcome to the GRO version of the STOL Tutor Orientation, Cliff Baker.

Click on the element that you want information on or press the "Auto" button for an automatic lesson.

Figure 3
GRO Orientation Screen of the STOL ITS



STOL> ACQUIRE ON HISD --> (ACQUIRE ON HISDR)

1. Please check HISD and following arguments.

2. Consider using the Glossary to look up the arguments for ACQUIRE.

3. The argument for ACQUIRE is HISDIR.

EXPLAIN There is 1 error in your entry. You entered "HISD" in position 3 of the command. The correct term for that position is "HISDIR".

Figure 4
Advanced Prepass Test/Tutoring Screen of the STOL ITS

new approach was developed in order that the tutor could be developed with a reduction in GRO FOT time. By providing the GRO FOT with a Glossary and Certification Tool, addition contact time was provided outside the formal structure of the knowledge engineering process.

Based on the experience of STOL ITS development, particular missions may be most willing to provide expert hours if they benefit directly from the ITS. The experts for a particular mission are in great demand, and it can be difficult to justify their time on non-mission related projects. If the ITS project can directly benefit mission training, then the FOT may be more likely to get involved.

FOT/SME Training Cycle Coincidence with ITS Development Cycle. There is a general mismatch between the development cycle of an ITS and the training cycle of an FOT. In addition to the actual number of hours required of a mission, it has proved important to consider when those hours are needed, and how the entire ITS development process will fit into the FOT's training needs. A persistent problem was the KE's need for specific types of experts or novices which do not coincide with their availability within the FOT. At the beginning of the FOT training cycle, there are usually several analysts who are spending up to six months gaining expertise on the specific spacecraft, but who are not sufficiently knowledgeable to serve as SMEs. The lesson here is that, for a new ITS, sufficient domain expert knowledge may not be available to develop an ITS which meets or precedes the needs of the FOT training cycle.

When it became evident that the STOL ITS would not be delivered until after the FOT had been certified, the STOL ITS development team decided to meet the needs of the GRO FOT by providing them with a certification tool that could be used in the normal FOT training cycle. That certification tool provides the STOL ITS development team with substantial information on trainee errors. In general, the ITS development teams needs to take a flexible and opportunistic approach to the development process.

Phase Activities. In developing an expert system, one approach, such as that recommended by the Expert System Development Methodology (ESDM), is to start with a key concept or cognitive function and develop a small system to evaluate the feasibility of the system. This is a useful first stage for some systems, but in the case of ITS development, it proved more efficient and compelling to first develop a prototype of the user interface. The prototype user interface is a much less abstract representation of the final system when compared with a small rule set that models some expertise. The prototype user interface can clearly demonstrate to users the intended functionality of the system and can provide early feedback on the proposed tutor capabilities.

## Project Reporting Lessons

The STOL ITS development project has generated a subset of the Expert System Development Methodology (ESDM) reporting requirements (CSC, 1989). ESDM structures expert systems development as an iterating process made up of the five stages shown in Table 2.

In general, ESDM has provided a good framework for the development of expert systems including ITSs. The STOL ITS development process spanned only three of the ESDM stages, and as such could not provide a complete evaluation of the ESDM process. This limited evaluation of ESDM has pointed out a number of issues. ESDM has been selectively employed for about nine months of the STOL ITS development cycle including most of the research prototype and a partial field prototype development. If the ESDM process had been fully implemented, approximately 10 to 12 reports would have been prepared. A significant amount of the development time is required to generate reports. One approach is to take the following ESDM recommended reports and consolidate them into two reports per stage preceded by a project initiation report and terminated with a final project report. The reports shown in bold were the most useful for the STOL ITS development cycle.

**Table 2. ESDM Stages and the Equivalent STOL ITS Stages**

**ESDM Stages**
Feasibility prototype
Research prototype
Field prototype
Production prototype
Operational prototype

**STOL ITS Development Cycle**
User interface development
Research prototype
Partial field prototype

(Asterisked reports are due for every stage of ESDM development):

- Concept and Project Initiation Report
- **Stage Project Management Plan***
- Prototype Design Report*
- Prototype Operations Guide*
- **Prototype T & E Report***
- **Knowledge Engineering Report***
- **Technology Transfer Report**
- Project Termination Report

In conclusion, ESDM provided a valuable framework, and its modification and partial automation would provide an extremely valuable tool for the management of KE processes.

## Project Cost and Schedule Control Effectiveness Lessons

Use of an Existing ITS Architectures, Tools, and Prototypers. One of the objectives of the STOL ITS project was to promote ways to reduce the development costs of the ITS. Use of an existing ITS architecture helped to greatly reduce the time of the STOL ITS development effort. The development costs for the STOL ITS were reduced by using NASA/JSC's general architecture. The use of that architecture reduced the ITS control coding requirements. Next, the user interface was prototyped in HyperCard which provided substantially reduced development times when compared with coding in C or some other programming language. Finally, CLIPS provided a more efficient expert system development environment than AI languages such as LISP or Prolog.

Figure 5 shows the estimated development type for the STOL ITS prototype to be about 1.5 man years. It was estimated that it would take about one-half a man year for each of the following three tasks: 1) Develop a prototype user interface, 2) Develop the CLIPS rules, and 3) Integrate the user interface with the rules .

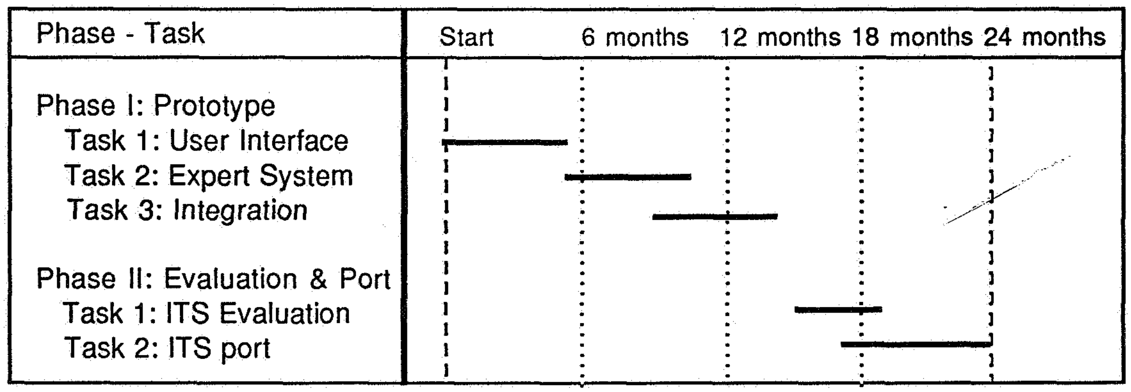


**Figure 5. Estimated STOL ITS Schedule in Man-Months (Seamster, 1989)**

The actual development time for the STOL ITS demonstration prototype came close to that, and is substantially shorter than the estimated 6 man years that it took to develop an operational version of NASA Johnson Space Center's Payload-assist module Deploys/Intelligent Computer Aided Training (PD/ICAT). There are some difficulties in comparing estimates for developing a prototype versus estimates for an operational system, but it is evident that the project costs were greatly reduced through the used of a general architecture and development tools. An even greater savings could be realized when using STOL ITS's modules and the general ITS prototyper in developing other ITSs for NASA command and control languages.

## Tools & Shells Effectiveness Lessons

The STOL ITS development process has brought together a number of existing tools and architectures to form a general ITS prototyper. This ITS prototyper has allowed for the relatively rapid development of an ITS. The general ITS prototyper is based on the following tools and applications: HyperCard, CLIPS, HyperCLIPS, and PD/ICAT. With some modification, the domain independent PD/ICAT rules were used as the basis of the control of the ITS prototyper.

General Need for Knowledge Acquisition Tools. During the KE process, a number of different software applications were utilized to perform the required data analyses. For example, a spreadsheet program was used to build the proximity matrices, and a statistical analysis program was used to run the cluster analysis. This process resulted in a large number of steps and substantial time spent entering and verifying data. As has been noted by other knowledge engineers, there is a great need for knowledge acquisition tools that can simplify the data collection and data analysis process.

In the early phases of the STOL ITS development, a brief review was made of existing knowledge acquisition tools with the conclusion that they are either too specialized or do not provide sufficient flexibility so that their results can be easily imported to other tools. This points out the strong need for a flexible and integrated set of knowledge engineering tools that can be used for collecting, analyzing, and reporting KE results.

HyperCard V.1.2.5. HyperCard, a Macintosh prototyping tool, was used to develop STOL ITS interfaces. It allowed for the rapid development of user interface prototypes by facilitating the design of buttons, dialogs, and interactive graphics, and it allowed for quick evaluation and modification of interfaces after review by potential STOL ITS end users. The major limitations of the 1.2.5 and earlier versions of HyperCard (screen surface size, graphics limitations) have been overcome by version 2.0 of Hypercard. SuperCard also provides these capabilities and may be a more flexible tool for ITS user interface prototyping.

CLIPS V 4.3 and HyperCLIPS V 1.0.2. The C Language Integrated Production System (CLIPS) was developed at the Johnson Space Center by the Artificial Intelligence Section for use in developing systems for the NASA Mission Control Center. CLIPS uses a forward chaining rule system with a syntax allowing free form patterns as well as single and multi-field variable bindings across patterns. CLIPS, implemented in the C language, is highly portable. The primary method of representing knowledge in CLIPS is a rule, a collection of conditions and the actions to be taken if the conditions are met. The expert system developer defines the rules which describe how to solve a problem, and the entire rule set makes up the knowledge base. CLIPS provides the inference engine which matches the rules to the current state of the system and applies the actions or consequents.

HyperCLIPS permits the execution of CLIPS rules from HyperCard. By integrating HyperCard and CLIPS rapid prototyping of knowledge-based expert systems may be accomplished. HyperCLIPS includes a number of programs which implements a specialized version of an XCMD. The CLIPS XCMD is named "ClipsX" and is designed to be as similar to the command line version of CLIPS. For this version of HyperCLIPS, the commands implemented are: clear, load, reset, and run. Data returned from CLIPS may be retrieved through the HyperTalk function, *the result*, after calling the run command. HyperCLIPS proved very useful in integrating the STOL ITS HyperCard user interface with the CLIPS rules. The primary limitation is the reduced number of CLIPS commands that are implemented. For example, it would be very useful if the CLIPS command unwatch all, could be passed prior to loading the CLIPS rules. This would save a substantial amount of time during the loading process. For prototyping purposes, HyperCLIPS V 1.0.2 has been very effective.

PD/ICAT. The STOL ITS architecture is based on a subset of the PD/ICAT rules which were subsequently modified into a general ITS prototyper. PD/ICAT was developed for NASA/JSC (Loftin, Wang, Baffes, & Hua; 1988), and was designed around a general ITS architecture that could be used for a number of different procedural training tasks. Based on the analysis of the PD/ICAT modules, the STOL ITS project utilized the rules from ERROR.CLP, LINKRULE.CLP, MANAGER.CLP, and SUPPORT.CLP as the primary control rules for the CLIPS portion of the general prototyping ITS architecture. These four modules were selected because they are relatively domain independent, and could provide the basic control for the ITS prototyper. The major form of modification of the PD/ICAT rules has been the replacement of the external functions that handle the output of the rules back to the user interface. The use of the PD/ICAT general architecture reduced the STOL ITS development time substantially and inspired the idea to develop a general ITS prototyper.

## CONCLUSIONS AND RECOMMENDATIONS

Subject Matter Expert Access. Missions may provide SMEs to support the KE process if they benefit directly from ITS development. The experts for a particular mission are in great demand, and it can be difficult to justify their time on non-mission related projects. Also, provision of development tools (Such as the Certification Tool) can increase SME contact time, and provide immediate support to the mission training program.

Prototyping. It proved efficient to develop early on a prototype of the user interface. This provides at least two strong advantages. First, the resulting prototype can be used to demonstrate the ITS to potential users and gain SME support and feedback. Secondly, it can be used in the early stages to more clearly define the requirements and capabilities of the system. Starting with a prototype of the ITS is a user-centered approach which can ensure that the final system will meet the needs and requirements of the trainees.

Documentation and Reporting. By consolidating ESDM reports, duplication of effort can be reduced, and the time spent preparing reports can also be reduced. For example, the management plan and design report could be combined to form an initial report for each stage. The operations guide and test and evaluation could be combined for a stage termination report. The knowledge engineering report proved to be a very useful report, but it is not clear that it is needed at every stage.

Cost and schedule control. Project costs were greatly reduced through the used of a general architecture and development tools. Greater savings could be realized when using STOL ITS's modules and a general ITS prototyper in developing other ITSs for NASA command and control languages.

Tools and shells effectiveness. A major part of the development of most expert systems, including Intelligent Tutoring Systems (ITS), is the prototyping phase. The development of knowledge-based systems requires prototyping for most of the stages, and consequently, the prototyping environment is critical to the success of such efforts. Prototyping has proven central to the development of the STOL ITS, and in the same way that STOL ITS benefited from NASA/JSC PD-ICAT development, future NASA ITS development could benefit from the STOL ITS KE process and tools. A general ITS prototyper could serve as a foundation for the development of future ITSs.

During the current STOL ITS effort, a number of general ITS prototyper requirements have been identified. That process could be expanded by the preparation of a formal requirements analysis for a NASA general ITS prototyper. These requirements would be extended to meet the needs of the NASA ITS community emphasizing the development of a prototyper which would allow for the quick development of ITS prototypes in the microcomputer environment. The new requirements would be analyzed to identify modifications and additions that need to be made to the STOL ITS prototyper. These modifications could then be implemented resulting in a general ITS prototyper to serve the ITS development needs of NASA.

GRO FOT STOL ITS Refinement. The current structure of the trainee model is based on that used in PD/ICAT, but the STOL ITS would be greatly improved if it were to adopt a more sophisticated model. The current trainee model has been developed as a database containing information about the trainee's understanding of specific productions. The STOL ITS tutoring relies on providing the trainee with practice of specific productions, and the trainee model keeps track of the productions' status. As the trainee makes mistakes with underlying productions, the trainee model database keeps track of those productions, and that data base can anticipate when the trainee might make an error on an advanced problem containing that production.

The STOL ITS demonstration prototype would make a good testbed for hypermedia application. The current version of STOL ITS is limited to the display of still pictures and text, and could be expanded to include voice, moving video images, and other interactive capabilities of hypermedia. For example, the STOL ITS Orientation Lesson shows all major elements of the GRO control system. In its current implementation, when the trainee selects an object, a static picture and/or text is presented to give additional detail. This type of tutoring would be enhanced if hypermedia were available to present a more realistic and animated view of the GRO operational environment.

## REFERENCES

Anderson, J. R. (1987). Production systems, learning, and tutoring. In D. Klahr, P. Langley, and R. Neches (Eds.), Production system models of learning and development (pp. 437-458). Cambridge, MIT Press.

Anderson, J. R., Conrad, F. G., & Corbett, A. T. (1989). Skill acquisition and the LISP tutor. Cognitive Science, 13, 467-505.

CSC (1989). Expert System Development Methodology User Guide. Prepared for NASA Goddard Space Flight Center, August 1989.

Eike, D. R., Seamster, T. L., & Baker, C. C. (1989). Applying Intelligent Tutoring System Technology to NASA Control Centers. (Draft) White Paper Report prepared for NASA Goddard Space Flight Center, May, 1989.

Eike, D. R., Seamster, T. L., & Truszkowski, W. (1989). Functional description of a command and control language tutor. Proceedings of SOAR'89: Third Annual Workshop on Space Operations Automation and Robotics (585-592). Houston, Texas.

Loftin, R. B., Wang, L., Baffes, P., & Hua, G. (1988). An intelligent training system for Space Shuttle flight controllers. Proceedings of the 1988 Goddard Conference on Space Applications for Artificial Intelligence (pp. 3-15). Greenbelt, MD.

Mitchell, C. M. (1989). Human-Computer Interaction in Distributed Supervisory Control. Semi-Annual Report prepared for NASA Goddard Space Flight Center, August, 1989.

Seamster, T. L. (1989). Cognitive Task Analysis: Techniques for the Development of Airborne Weapons Training. Final Report prepared for the Data Systems Engineering, Oak Ridge National Laboratory.

Seamster, T. L. (1989). Functional Requirements Document: Systems Test and Operations Language (STOL) Intelligent Tutoring System (ITS). (Draft) Technical Report prepared for NASA Goddard Space Flight Center, August, 1989.

Seamster, T. L. (1990). Architecture Review: General Intelligent Tutoring System (ITS) Prototyper. Draft Report prepared for NASA Goddard Space Flight Center, April, 1990.

Seamster, T. L. (1990). Systems Test and Operations Language (STOL) Intelligent Tutoring Systems (ITS) Knowledge Engineering Report. (Draft) Technical Report prepared for NASA Goddard Space Flight Center, July, 1990.

Seamster, T. L. (1990). Architecture Review: General Intelligent Tutoring System (ITS) Prototyper. (Draft) Technical Report prepared for NASA Goddard Space Flight Center, April, 1990.

# Space Communications Artificial Intelligence For Link Evaluation Terminal (SCAILET)

Anoosh Shahidi
Sverdrup Technology, Inc.
Lewis Research Center Group
2001 Aerospace Parkway
Brook Park, Ohio 44142

## 1. INTRODUCTION

A software application to assist end-users of the Link Evaluation Terminal (LET) for satellite communications is being developed. This software application incorporates artificial intelligence (AI) techniques and will be deployed as an interface to LET. The high burst rate (HBR) LET provides 30 GHz transmitting/20 GHz receiving, 220/110 Mbps capability for wideband communications technology experiments with the Advanced Communications Technology Satellite (ACTS). The HBR LET and ACTS are being developed at the NASA Lewis Research Center in Cleveland, Ohio. The HBR LET can monitor and evaluate the integrity of the HBR communications uplink and downlink to the ACTS satellite. The uplink HBR transmission is performed by bursting the bit-pattern as a modulated signal to the satellite. By comparing the transmitted bit pattern with the received bit pattern, HBR LET can determine the bit error rate (BER) under various atmospheric conditions. An algorithm for power augmentation will be applied to enhance the system's BER performance at reduced signal strength caused by adverse conditions.

The HBR LET terminal consists of seven major subsystems (Fig. 1):

- Antenna subsystem
- Radio frequency (RF) transmitter subsystem
- RF receiver subsystem
- Control and performance monitor (C&PM) computer subsystem
- Local loopback subsystem at RF
- Modulation and BER measurements subsystem
- Calibration subsystem

The C&PM computer controls and monitors all the other subsystems through an IEEE488 interface. HBR LET experiments with the ACTS satellite will be initiated by users through the C&PM experiment control and monitor (ECM) software. The ECM software was developed on a Concurrent 3205 minicomputer in FORTRAN, which provides the end-user with the following capabilities:

- Individual instrument control
- Interactive interface used to communicate with the digital ground terminal
- Ability to conduct BER measurements

- User-controlled data acquisition

Programming scripts, defined by the design engineer, set up the HBR LET terminal by programming subsystem devices through IEEE488 interfaces. However, the scripts are difficult to use, require a steep learning curve, are cryptic and are hard to maintain. The combination of the learning curve and the complexities involved with editing the script files may discourage end-users from utilizing the full capabilities of the HBR LET system. In the following sections, I describe an intelligent assistant component of SCAILET that addresses critical end-user needs in the programming of the HBR LET system as anticipated by its developers. We will then take a close look at the various steps involved in writing ECM software for a C&PM computer and at how the intelligent assistant improves the HBR LET system and enhances the end-user's ability to perform the experiments. (Although a hypertext documentation module plays an important role in familiarizing end-users with all the LET HBR subsystems, the description of this module is beyond the scope of this paper.)

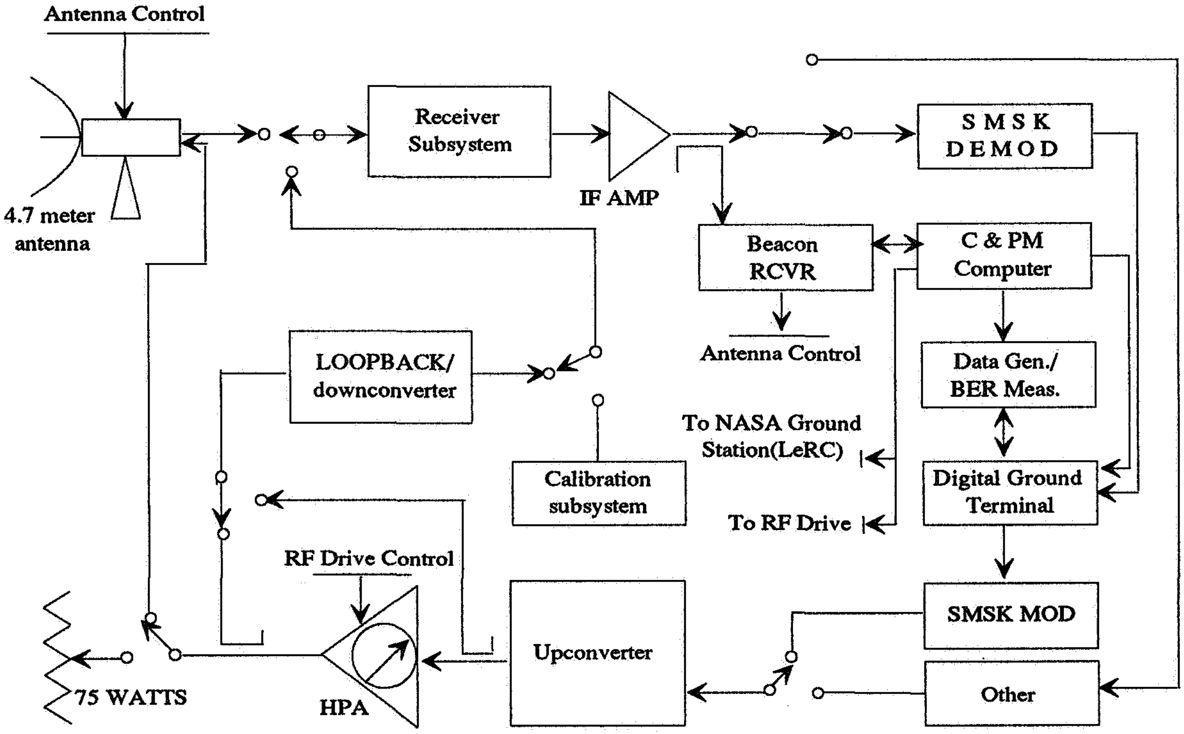


Figure 1. HBR LET Block Diagram

## 2. DILEMMA OF SOPHISTICATED INTERFACES

The fundamental dilemma in designing practical software is how to provide more power to the user without sacrificing ease of use. By designing intelligent interfaces the gap between the novice user who is a domain expert and the software can be bridged. In my research, I view the user as a "planner." Planning is a problem-solving technique, the process of finding a sequence of steps to accomplish some goal. In the system described here the computer user manipulates knowledge structures called *plans*. Plans are bundles of knowledge about the standard subtasks in a domain and are designed and organized from a typical user's point of view.

There are many different approaches that address planning in artificial intelligence. Two major approaches are hierarchical planning (e.g., Sacerdoti, 1974) and script-based planning (Shank, and Abelson, 1977). Most traditional planners try to generate a plan of action for a specific task in a domain. In contrast, my planning system is designed to provide a framework in which executable forms of domain tasks can be specified by using a planning hierarchy. A planning hierarchy provides a view of a procedural specification (a sequence of actions) that achieves a domain task which includes an explicit notion of levels of detail. Therefore, the novice user can be supplied with a portfolio of functionality - predefined high-level plans that omit many details - for using the software in each task domain.

| | |
|---|---|
| Level 1: | John Doe's BER.Back to Back terminal Only |
| Level 2: | Generic BER, Generic Loopback, Generic Calibration |
| Level 3: | ECM Software |
| Level 4: | FORTRAN |

**Figure 2.** Implementation Hierarchy for SCAILET Software. Each level represents a specialization built out of the primitives provided at the next lower level.

Figure 2 uses the example of SCAILET software to illustrate the application of a planning hierarchy to intelligent interfaces. The SCAILET development environment is a Gateway 486 personal computer and uses the Choreographer graphical user interface tool. SCAILET communicates with the Concurrent computer through a TCP/IP link. At the top level there is experiment-specific software for a particular project. The second layer is general categories of software (e.g., BER, calibration,and loopback). The third layer is the ECM software developed on the Concurrent computer. Finally, the fourth layer is FORTRAN. Each layer has its own "primitives," which encapsulate a sequence made up from primitives at the next level down.

I divide my computer users into three categories: domain expert, computer novice (DECN); domain novice, computer expert (DNCE); and domain expert, computer expert (DECE). For instance a DECN would be an experienced satellite operator who wants to use HBR LET to test video transmission on the ACTS satellite but has no previous experience on the Concurrent computer. A DECE could be an experienced communications engineer who uses our software applications to get data. A DNCE could be a computer scientist who knows how to use software applications but knows nothing about communications.

Currently, a researcher (DECN) begins with the ECM software layer (level 3); the two upper levels only exist in his or her mind. Occasionally, the only DECE has enough time to actually create a version of level 2 or level 1 for use by a DECN. More likely, the DECN must learn the ECM software's commands - which are the primitives for level 2 - that are necessary to perform actions at the higher levels. The highest level, which is customized to the needs of a specific user, can only be built by a DECE user who is proficient in levels 2, 3, and 4 and can make proper use of the primitives at each level. The point is to provide an environment that allows these higher levels - levels 1 and 2 - to be created by a DECE for use by a DECN.

## 2.1 Barrier Between DECN and DECE

This general model of software use represents domain tasks by nodes at the top level of a tree. These tasks are gradually "unpacked" by each subsequent level below, so that at some level the nodes describe a sequence of how the task domain is accomplished by using the ECM software. The specific goal of this research is to build a system that allows a DECN to start at level 1 and then facilitate his or her movements through the first 3 levels.

In using a general-purpose application a DECN user must first learn the commands and the macro language of the software. This step lacks context; the DECN is forced to deal with an abstract formalism divorced from his or her expertise. In the context of this example, typically force researcher John Doe (DECN) to use the ECM package. After mastering the ECM package's commands, John Doe has to mentally create intermediate plans that are primitive relative to his domain of expertise. Using these intermediate plans, John Doe must then create overall plans that are specific only to his situation.

The operators at level 1 are overall goals - they resemble goals in the task domain; while the operators at level 3 are data specialized goals - they are the commands of a general purpose applications package. The goal of this research is a framework and an environment in which the general purpose software can be specialized by a DECE, allowing the DECN to draw analogies between the task domain and his or her own knowledge at the overall goal level and initially avoiding abstract formalisms of the data specialized level.

## 2.2 STANDARD APPROACH TO BARRIER

Most intelligent interfaces that have tried to break down this barrier attempt to monitor user actions and try to infer an overall plan (e.g., Johnson, 1986). These systems are usually partial matching schemes, based on a plan catalog. The inferences made by partial matches allow the system to correct mistakes, to complete actions, and to infer "higher" plans. Since these systems mainly monitor low-level user actions and infer higher level plans, I refer to these intelligent interfaces as plan-matcher systems. One drawback of these systems is that matching always requires a detailed understanding of the user's goals - something that is not typically available. Also, since a novice's actions are erratic, plan-matchers that try to infer the overall plan from a novice's actions are often brittle. In the example a plan-matching system would have to monitor the ECM package primitives used by the user, refer to a standard set of novice goals, and infer plans that the novice is using to accomplish those goals. In most domains this a very difficult approach.

Besides the implementation difficulties, the overall approach, of necessity, focuses on helping a novice select a set of primitive steps in using the software. In contrast, the plan language of SCAILET will focus on the organizational and intermediate steps that allow the user to achieve an overall plan in the software domain. The actual primitive steps provided by the software package typically are never used by a novice. For instance, an engineer knows that he wants to transmit and receive video signals to and from the ACTS satellite but does not know the first thing about how to use the ECM application and how to structure his domain knowledge to achieve this high-level goal. Given the standard knowledge that has to be in the system explicitly, the traditional bottom-up approach makes the engineer work much harder than necessary. The traditional approach makes the engineer learn the low-level ECM commands and then tries to infer that all the engineer wants to do is to transmit and receive a signal. Since explicit knowledge for this task already exists, SCAILET plan language gives the engineer a transmitting and receiving plan directly and lets him specialize it. The SCAILET approach is explained in detail in the following section.

## 3. THE SCAILET PLAN LANGUAGE

### 3.1 Related Work

Programmer's Apprentice (PA) Project (Waters et. al., 1985) was one of many automatic programming research projects that had the goal of improving programmer productivity by developing tools based on AI techniques. This project also studied human-problem solving behavior by using the programming domain. The long-term goal of the PA Project has been to develop a theory of programming (i.e., how expert programmers understand, design, implement, verify, modify, and document programs). Although the Emacs knowledge-based editor (KBEmacs) in the PA project falls well short of the long-term goal, it offers interesting insight into the task of program construction. This knowledge-based editor is tightly integrated with a standard Emacs-style editor. This integration allows the programmer to freely intermix knowledge-based program editing with text-based and syntax-based program editing.

The most dramatic development in programming has been the development of high-level languages. Now, automatic programming is attempting to perform middle-level programming decisions automatically, and hence is, bringing about a second improvement. AI techniques make it possible to represent knowledge about programming in general and use this knowledge to understand particular programs. There are three main ideas that were implemented in KBEmacs:

(1) The assistant approach: Since fully automated programming is too hard to implement with what is now known, Waters et al., (1985) decided on an assistant approach, a division of labor between the programmer and his or her assistant, the KBEmacs. The assistant will take

care of the low-level operations, in order to make a human programmer more productive. This approach can also serve as a research ground for fully automated programming.

(2) Cliche: A cliche is a standard method of dealing with a task, a lemma or partial solution. Cliches are aggregates of code that achieve a stereotypical operation (e.g. searching a one-dimensional structure). When a cliche is used, it is instantiated by filling in the roles (i.e., input or output variables) with appropriate computational tasks. This creates a cliche that is specialized to the task.

An important aspect of cliches is reuse. Once something has been thought out and given a name, it can be used as a component in future thinking. Cliches provide an appropriate vocabulary for relevant intermediate and high-level concepts. Both man and machine are limited in the complexity of the lines of reasoning they can develop and understand. In order to deal with more complex lines of reasoning, intermediate-level vocabulary can be used that summarizes parts of the line of reasoning.

```
cliche EQUALITY_WITHIN_EPSILON
      Primary roles X, Y, EPSILON;
      comment "determines whether {the x} and {the y}
            differ by less than {epsilon}";
      constraints
            DEFAULT ({the epsilon}, 0.00001);
      end constraints;
begin
      return abs({the input x} - {the input y}) < {the epsilon};
end EQUALITY_WITHIN_EPSILON;
```

Figure 3. The Cliche EQUALITY_WITHIN_EPSILON From the PA Project.

A corollary of the cliche idea is that a library of cliches can be viewed as a machine-understandable definition of the vocabulary programmers use when talking about programs. In KBEmacs a large portion of the knowledge that is shared between the programmer and the computer is in the form of a library of algorithm cliches.

Figure 3 is a simple example of a cliche in KBEmacs. The EQUALITY_WITHIN_EPSILON cliche compares two numbers and returns a boolean value that specifies whether or not the numbers differ by less than a given epsilon. The roles that must be filled are X and Y, which are numbers to be compared with one another. A constraint is used to specify a default value for epsilon, but the user can specify his or her own. When cliches are used, they can be specified as an indefinite noun phrase (e.g. "an EQUALITY_WITHIN_EPSILON of A and B"). The primary roles definition lists the roles that must be specified and the order in which they must be specified. When this cliche is instantiated in a program the roles in braces(i.e., {.....}) are replaced with actual numbers. For example, if two values, A and B, were passed to this cliche, the comment would read " `determines whether A and B differ by less than 0.00001.`"

(3) Plans: Many AI systems make use of plans as a way of dealing with complex operations. As a representation, plans can deliberately ignore some aspects of a problem in order to make it easier to reason about other aspects of a problem. Plans are designed to represent two kinds of information in KBEmacs: the structure of the particular programs, and the knowledge about cliches. The two basic operations performed by KBEmacs are simple reasoning about programs (i.e., the source of data flow) and combining cliches together to create programs. The plan formalism is particularly designed to handle these operations (e.g., explicit arcs to show data flow make it easy to determine the source of the data). The user can construct programs by specifying the cliches that will be used within that program and specializing the roles of those cliches.

The plan formalism abstracts away from the syntactic features of a programming language and allows the programmer to focus directly on the semantic features of a program. This also has the added advantage of making the internal operations of KBEmacs language independent.

The major advantages of KBEmacs claimed by Waters et al. (1985)are:

• Programs can be constructed more quickly.

• Since programmers are limited in the amount of code they can produce per day, it is more productive to specify which cliches and what roles are used in a program than to write the code for each program. Since cliches are intended to be reused, the time invested making cliche libraries is worth while.

• A program built out of cliches is more reliable.

The major disadvantage of programming in cliches claimed by Waters et al. (1985) is that to get the full benefit of cliches, the programmer has to think in terms of them as much as possible.

KBEmacs was only a research prototype fraught with bugs. It was a 40,000 line Lisp program that had only a dozen cliches. Designing exact cliches to be used was a lengthy task.

The idea of plans as a method of program construction was studied by Soloway and Ehrlich (1985). Soloway's empirical studies suggest that expert programmers use two types of programming knowledge: (1) programming plans, which are generic program fragments that represent stereotypical action sequences in programming; and 2) rules of programming discourse, which capture the conventions in programming and govern the composition of the plans into programs. Experts seem to have a portfolio of these plans that can be used in problem solving.

As a representation, plans can deliberately ignore some aspects of a problem (i.e., syntactic structures) in order to make it easier to reason about other aspects of a problem (e.g., program design). One of the most powerful ideas in AI is the idea of a representation shift (shifting from text representation to plan representation). Initially, the novice would have to learn which language structures (i.e., to use syntactic choices) and then code those structures correctly (i.e., syntactic structures). Kurland et al. (1986) see the stage of learning the syntactic structures of a programming language as a barrier that poses significant difficulty for the novice. By removing or reducing the syntactic structures, it possible to reduce the cognitive load on the novice and allow him or her to focus on design issues. It is important for the novice to build a model of the computational process, so that he or she can predict actions and debug programs. When the cognitive load resulting from the syntactic structures of the language is reduced, the novice can focus on the flow of control and build his or her computation model.

### 3.2 The SCAILET Approach to the Barrier between DECN and DECE

The SCAILET approach provides a plan language with which to specify the layers of plans that make up a hierarchy for a range of tasks and goals. The plan language provides a knowledge-structuring scheme that will house a DECE's understanding and structuring of domain knowledge in a form usable by a DECN. Although this is much like an object hierarchy, but it includes enough information for the plan language to help a DECN use the hierarchy at multiple levels.

The DECE users will be provided with an environment where they can build domain knowledge into layers, so that the coded knowledge will not just be visible to another programmer, but will also be usable and visible to a DECN who wishes to achieve domain tasks. The top level of the SCAILET application will be goal driven: A DECN will be able to draw analogies between this level and domain tasks and will then use the top-level plans to achieve domain tasks. The application will be provided to the DECN as a set of plans that represent the tasks and subtasks which are of interest to him or her.

### 3.3 Current ECM Program Structure

A typical ECM program contains the following modules:

(1) Instrument definition software
- (a) Specifies instrumentation to be included in the experiment or test
- (b) Provides initial configuration and control parameters for each instrument

(2) Sequence definition software
- (a) Encodes the experiment as a sequence of commands
- (b) Distinguishes between the main sequence and subsequences

(3) Sequence execution software

BER sequence commands are given in the appendix. As you can see, it is very cryptic and requires a steep learning curve.

### 3.4 Formalizing SCAILET Programming Plans

The plan language system provides HBR LET system designers with an easy and structured way to construct plans as bundles of programming code with data and control links to other plans. Using this mode, a designer will be able to build his or her portfolio of programming plans that can be reused in various problems.

The plan language has the following design goals:

(1) Support the system designers in developing a portfolio of plans.

(2) Support the use of a plan-like composition of programs.

(3) Allow a distinct mechanism for data flow between plans.

(4) Allow a distinct mechanism for the flow of control within a program.

The plan formalism is based on object-oriented programming. For each plan there is a class that specifies the local data and operations of that plan. Plan class can then be specialized into instances, each with its own copy of local data. The DECE can sequence these instances into a particular execution order. Each plan object consists of four parts:

(1) PARENTCLASS - This is a hierarchical link that is part of the inheritance of the plan language.

(2) SLOTS - Each plan can have zero or more slots that specify data or plan links. Data slots are used to store data that are used during the execution of a plan. A plan slot is viewed as a component within the owner plan.

(3) INITIALIZATION - This part contains executable code that is performed once when control first flows through the plan.

(4) EXECUTION - This part contains executable code that is performed whenever control flows through the plan.

### 3.5 An example: BER testing

SCAILET plan language envelops procedural commands of ECM software in a declarative subplan that can be referenced by the end-user. Using ECM software, the user can conduct BER testing, loopback, and calibration. Figure 4 is a specialized BER plan language interface with three subplans for create files, instrument definition, and develop sequence. The structure of the program is intuitively apparent. ECM commands are bundled in subplans at the lowest level. The user does not have to know specific ECM commands. He or she only has to choose among various subplans. I intend to deploy SCAILET with a large number of typical plans. The user can then specialize or modify any of the SCAILET plans for his or her specific needs. Subplans know their parent plans, and hence tell the user when they are being misused. The system can also detect if there are any initial values missing for any subplans. Plans and subplans are designed by using graphical screens.

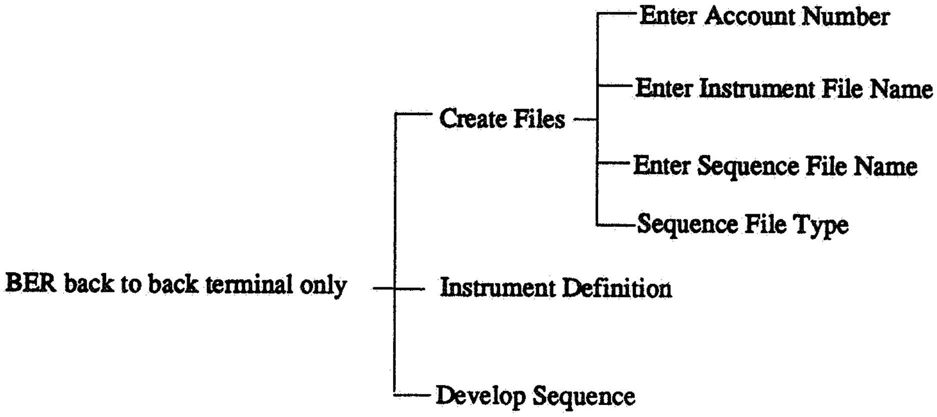


**Figure 4. Specialized BER Plan Language Interface With Three Subplans (The user can click on a plan and "open up" its subplans).**

## 4. FUTURE DIRECTION

By 1992, a complete portfolio of typical plans will be developed by using the system developers. Since the system will also have a completed hypertext documentation system, links between the plan language subplans, and their corresponding documentation will be established.

## ACKNOWLEDGMENTS

This project is being developed at and funded by the Space Electronics Division of the NASA Lewis Research Center. I would like to thank Mr. Edward Petrik of NASA Lewis for his support and direction on this project. I would also like to thank Mr. Rich Rienhart of Analax Corp., the programmer of the ECM software, and Mr. Rich Schlegelmilch of the University of Akron for his system support, and his contributions to this project and to the hypertext documentation module.

## REFERENCES

Johnson, W.L. (1986). "Intention-Based Diagnosis of Novice Programming Errors". Morgan Kaufmann Publishers, Los Altos, CA

Kurland, D.M., Pea, R.D., Clement, C., and Mawbey, R. (1986). "A Study of Development of Programming Ability and Thinking Skills in High School Students". *Journal of Educational Computing and Research*, 2(4), pp. 429-458. Baywood Publishing.

Sacerdoti, E. D. (1974). "Planning in a Hierarchy of Abstract Spaces". *Artificial Intelligence* 5:115-135.

Shank, R. and Abelson, R. (1977). "Scripts Plans Goals and Understanding." Lawrence Erlbaum Associates, Publishers, Hillsdale, NJ

Soloway, E., Ehrlich, K., (1985) "Empirical Studies of Programming Knowledge". *Readings in Artificial Intelligence and Software Engineering*. Rich, C., Waters, R.C. (Eds) Morgan Kaufmann.

Waters, R.C. (1985) "The Programmer's Apprentice: A Session with KBEmacs". *Readings in Artificial Intelligence and Software Engineering*. Rich, C., Waters, R.C. (Eds) Morgan Kaufmann.

```
M1:SDALET.DOC/111
SEQUENCE ARRAY  SDA(24.500)

INT - 1  SEQUENCE COMMAND NUMBER +LABEL #         ACTION NUMBERS
          - 100000  SINGLE EXECUTION               21   SET A PARAMETER
          - 200000  START LOOP SEQUENCE            60   ZERO POWER METERS
          - 299999  END LOOP SEQUENCE              61   STEP A PARAMETER
          - 300000  CALL SUB-SEQUENCE             101   WAIT
          - 399998  END SUB-SEQUENCE              102   GOTO STATEMENT
          - 900000  BEGIN STOP SEQUENCE           110   CHECK A PARAMETER
          - 999997  END SEQUENCE COMMAND          301   START DATA GENERATOR
INT - 2 NUMBER OF TIMES TO EXECUTE LOOP           302   STOP DATA GENERATOR
INT - 3 NUMBER OF STEPS IN LOOP SEQUENCE          303   STOP DATA CHECKER
INT - 4 ACTION TO BE TAKEN                        311   SET NUMBER OF ERRORS FOR GENERATOR
                                                  313   PERFORM BER MEASUREMENT
INT - 5 CHECK PARAMETER - ARRAY SLOT OF INST      315   DGT COMMAND
        SET PARAMETER - ARRAY SLOT OF INST
        STEP PARAMETER - ARRAY SLOT OF INST
        START DG - ARRAY SLOT OF INST
        STOP DG - ARRAY SLOT OF INST
        STOP DC - ARRAY SLOT OF INST
        SET ERRORS -  ARRAY SLOT OF INST

INT - 6 SET PARAMETER - WAVETEK -> MARKER EDGE 1-RISING 2-FALLING
        WAIT - TOTAL TIME IN SEC
        SET ERRORS - NUMBER OF ERRORS TO SEND TO DG

INT - 7 SET PARAMETER - WAVETEK -> OPTION 0-15
                        BEACON RECEIVER -> OPTION 1-6
        STEP PARAMETER - INCREMENT / DECREMENT FLAG  1 UP  2 DOWN

REAL -8 SET PARAMETER - VALUE / PS VOLTAGE
        STEP PARAMETER - INC VALUE
        CHECK PARAMETER - LOW LIMIT


REAL -9 CHECK PARAMETER - HIGH LIMIT

INT -10 FUNCTION TO BE PERFORMED (CHAR)

INT -11 CHECK PARAMETER - LABEL TO GOTO IF UNDER RANGE
        GOTO - LABEL NUMBER TO GOTO
        END - TOTAL NUMBER OF SEQUENCE COMMANDS

INT -12 CHECK PARAMETER - LABEL TO GOTO IF OVER RANGE

CHAR-13 SUB-SEQUENCE - FILENAME
    -17 DGT COMMAND  - 20 CHARACTER COMMAND
        END SEQUENCE - SDA FILE NAME
```

```
INT -18 SET PARAMETER - TOGGLE SWITCH -> INPUT
                        WAVETEK MARK   -> CHANNEL MODE 0-12 OF WTPM TO SET
                        FREQUENCY -> CHANNEL MODE 0-12 OF WTPM TO SET
        CHECK PARAMETER - MODE OF WTPM TO CHECK 0-12
        START DG - DATA TYPE
        PERFORM BER - # OF MEASUREMENTS PER EB/NO

INT -19 SET PARAMETER - TOGGLE SWITCH -> OUTPUT
        START DG - DATA RATE
        PERFORM BER - DG NUMBER

INT -20 PERFORM BER MEASUREMENT - TIME TO WAIT BETWEEN READINGS
        START DG - DESTINATION



INT -21 START DG - MODEM RATE
        SET PARAMETER - EBNO -> MODEM RATE

INT -22 START DG - BURSTED

INT -23 START DG - CONTINUOUS

INT -24 SET PARAMETER - SWITCH SETTINGS = BIT POSITIONS 1,10


WAVETEK PARAMETER OPTIONS/MODE          WAVETEK MODES            BEACON RECEIVER OPTIONS

 0 - FREQUENCY                           0 -                     0 - FREQUENCY
 1 - REFERENCE DELAY                     1 - CW A&B              1 - 2ND LO VFO
 2 - CURSOR DELLAY                       2 - CW A                2 - VIDEO ATTENUATION
 3 - START DELAY                         3 - PEAK A&B            3 - SIG STR OFFSET
 4 - WINDOW DELAY                        4 - CW B                4 - CARRIER INDICATOR
 5 - REFERENCE POWER                     5 - GRAPH A             5 - SIG STR SLOPE
 6 - PULSE RISE TIME START               6 - PEAK A
 7 - PULSE RISE TIME END                 7 - GRAPH B
 8 - PULSE FALL TIME START               8 - PEAK B
 9 - PULSE FALL TIME END                  9 - MARKER A
10 - PULSE WIDTH TIME START             10 - MARKER B
11 - PULSE WIDTH TIME END               11 - PULSE A
12 - MARKER 1                           12 - PULSE B
13 - MARKER 2
14 - MARKER 3
15 - MARKER 4
```

# Knowledge Repositories for Multiple Uses

Keith Williamson, Patricia Riddle
Advanced Technology Center
Boeing Computer Services
P.O. Box 24346, MS 7L-64
Seattle, WA 98124-0346
(206) 865-3281

January 22, 1991

## Abstract

*In the life cycle of a complex physical device or part, for example, the docking bay door of the space station, there are many uses for knowledge about the device or part. The same piece of knowledge might serve several uses. Given the quantity and complexity of the knowledge that must be stored, it is critical to maintain the knowledge in one repository, in one form. At the same time, because of the quantity and complexity of knowledge that must be used in life cycle applications such as cost estimation, re-design and diagnosis, it is critical to automate such knowledge uses. For each specific use, a knowledge base must be available and must be in a form that promotes the efficient performance of that knowledge base. However, without a single source knowledge repository, the cost of maintaining consistent knowledge between multiple knowledge bases increases dramatically; as facts and descriptions change, they must be updated in each individual knowledge base.*

*We have developed a use-neutral representation of an hydraulic system for the F-111 airplane. We demonstrated the ability to derive portions of four different knowledge bases from this use-neutral representation: one knowledge base is for re-design of the device using a model-based reasoning problem solver; two knowledge bases, at different levels of abstraction, are for diagnosis using a model-based reasoning problem solver; and one knowledge base is for diagnosis using an associational reasoning problem solver. We have shown how updates issued against the single source use-neutral knowledge repository can be propagated to the underlying knowledge bases.*

## 1 Problem

In the life cycle of a complex physical device or part (e.g., the docking bay of the space station) there are many uses for knowledge about the device or part, about its structure, function, materials, component parts and so on. Such uses might include product definition (i.e., design and manufacturing process planning), testability analysis and test construction, cost estimation, producibility studies, diagnosis of breakdowns, and re-design of features. In an effort to become more competitive, corporations are automating such knowledge uses. For each particular use, a knowledge base must be available and must be in a form that promotes efficient performance of that system. This has resulted in duplicate knowledge in multiple knowledge bases, where a particular piece of knowledge serves multiple uses.

Given the quantity and complexity of the knowledge that must be stored, it is critical to maintain the knowledge in one repository. Without a single source repository, the cost of maintaining the knowledge required by multiple knowledge bases is increased dramatically; as facts and descriptions change, they must be updated in each individual system. It is virtually impossible to maintain synchronicity among the knowledge bases that require the same knowledge. Without a single source repository, the costs of other maintenance tasks (e.g., knowledge verification and validation) are not easily shared across knowledge bases. Finally if there is no common knowledge repository, the development of new knowledge bases must essentially be done from scratch each time.

To establish a single source knowledge repository, we cannot simply borrow representations from today's knowledge bases. In current knowledge base technology, the use of the knowledge determines its representation. The knowledge encoding is tailored for efficiency in performing the particular task at hand. However, once the knowledge representation is customized for a single use, it is difficult or impossible to apply it to some other use.

In order to maintain knowledge about a complex device in one knowledge repository for multiple uses, a use-neutral representation is required. In order to automate the use of that knowledge, it must be made available to many different problem-solvers and must be put in a form that promotes their efficiency. The need to manage and access single source information for multiple uses places critical requirements on the kind of knowledge that must be captured and the way the knowledge is represented in the repository.

Section 2 gives an overview of the objectives of our project. In section 3 the more detailed issues which arise from this research are discussed. Section 4 gives a survey of related research. In section 5 the initial focus of our project is given. Section 6 gives the progress made to date. In section 7 the future research directions are given. Section 8 concludes.

## 2 Project Objectives

The overall objective of this project is to develop a methodology for sharing knowledge between several knowledge bases which have different uses (e.g., design and process planning). This methodology consists of a single source use-neutral knowledge repository, and the ability to down-load this knowledge into different knowledge bases tailored for specific tasks. Specific objectives of the project include:

1. Demonstrate tools for transforming and downloading repository knowledge into knowledge bases that can be used by multiple problem solvers, that support the pre-defined range of life-cycle uses.
2. Allow the knowledge repository to be updated, and demonstrate the capability of automatically propagating these updates to the pertinent knowledge bases so as to preserve correctness.
3. Develop a methodology for creating a single source, use-neutral knowledge repository from a set of related use-specific knowledge bases.
4. Demonstrate the ability to automatically generate a first cut at a brand-new knowledge base from the knowledge repository.
5. Evaluate and establish knowledge representation principles for modeling physical devices/parts that support a pre-defined, but extensible, set of uses throughout the device/part's life cycle.
6. Establish requirements on the completeness and consistency of the single source, use-neutral knowledge repository imposed by the pre-defined range of life-cycle uses and develop tools to ensure those requirements are met by the single source model.

## 3 Issues Raised

The major benefits of such an approach are derived from three sources: from physically sharing knowledge, from updating the repository, and from adding new knowledge bases. From physically sharing knowledge, the knowledge bases have increased consistency with each other and a lower combined management cost and verification and validation (V&V) cost than if supported separately. From updating the repository and propagating the updates to the individual knowledge bases, the knowledge bases have a lower combined update maintenance cost and it is easier to maintain correctness and configuration control of the system. From adding new knowledge bases, this approach enables the identification of inconsistencies in pre-specified KBs and the partial automation of the generation of new KBs.

In this section, the issues raised in achieving these benefits are explored. In later sections, the focus of our project within these issues and our initial progress will be described. The global structure of the ultimate system is shown in Figure 1. One important aspect of this structure is that it allows the transformations (i.e., the gateway) to access knowledge from several different knowledge stores. This is very important with regards to the Boeing Company. There are knowledge stores for Boeing Corporate Standards which are accessible by vendors and proprietary knowledge stores which are not. These knowledge stores might be stored and accessed separately for security reasons, but their knowledge might need to be combined to create useful knowledge base rules to be used within Boeing. To a system accessing the knowledge repository, the knowledge should appear seamless. Even with nonproprietary knowledge stores, the information might be stored separately (e.g., some in Auburn, Washington and some in Wichita, Kansas). This information must be accessed directly from these different sources so as to maintain the single source model. The ultimate goal is a distributed single source use-neutral knowledge repository.

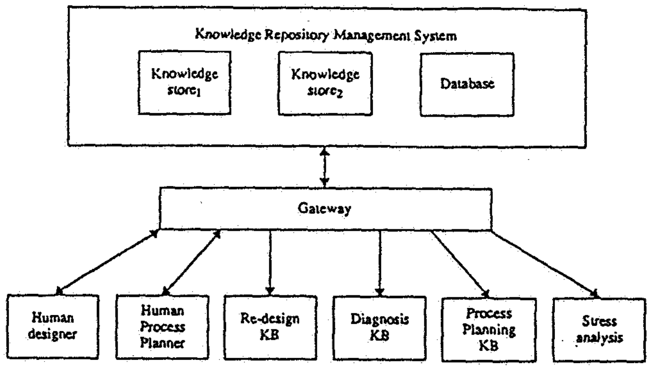


Figure 1: Single Source Repository

Another aspect of this model is that the language used in the use-neutral knowledge repository should have a sufficient expressive adequacy to allow new knowledge bases and their related transformations to be added to the repository later on. The knowledge in the single source repository is not use-neutral in the broadest sense. The knowledge must be represented to serve the "use" of transmitting the pertinent knowledge to multiple knowledge bases. But the knowledge representation used in the single source repository is independent of the specific uses of those knowledge bases, so it will be referred to as use-neutral throughout this paper.

Transformations which down-load knowledge must preserve the aspects of the model which are important for this knowledge base use (i.e., they must be behaviorally equivalent). In certain cases when a knowledge base requires all the details of the model this transformation will be the standard notion of equivalence (i.e., an isomorphic transformation). But in other cases transformations will either abstract away certain knowledge which is not necessary for this knowledge base or approximate knowledge for efficiency reasons. In these cases the transformation will only be behaviorally equivalent. The transformation will be equivalent only with respect to the desired behavior (i.e., a homomorphic transformation). For instance, if stress analysis is done, certain knowledge about the color of the paint or the smoothness of the finishes are not relevant knowledge with respect to the stress analysis.

The ultimate transformational system we envision is a partial-order of transformations (i.e., most likely a forest) from one representation to another. The reformulation from any one representation into another representation (i.e., either the use-neutral representation, a user representation, or the representation associated with a specific problem solver) is a path through this partial-order of transformations. This will allow a maximal amount of reuse of the transformations. Figure 2 illustrates this idea.

The KRMS (Knowledge Repository Management System) must deal with issues of V&V (e.g., consistency and redundancy) and configuration management across multiple knowledge stores. Configuration management includes keeping a history of any updates to the use-neutral knowledge repository so that the version of the knowledge given to a certain knowledge base or person at a certain time can be reconstructed. There is a tradeoff between applying V&V directly to the knowledge repository and applying it to each knowledge base separately. Some tests (e.g., consistency checking) might be best applied to the knowledge repository

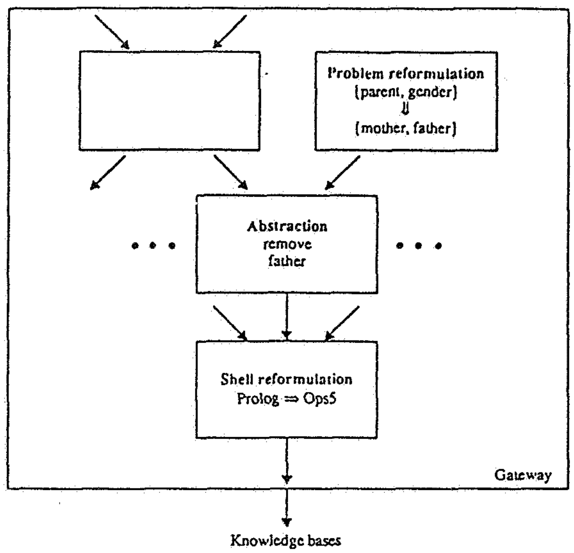


Figure 2: The Gateway Transformations

to assure synchronicity between all the knowledge bases and to avoid the cost of testing each knowledge base separately. Other tests (e.g., functionality testing) might be best applied to the task-specific knowledge base. It is not clear how to test the functionality of the knowledge repository directly.

The knowledge down-loaded to the task-specific knowledge bases must be consistently updated when an update is performed on the knowledge repository. There could be problems with this updating process within either the knowledge repository or the transformations. The representation of the knowledge in the knowledge repository might not be expressive enough to handle the update. The transformations might no longer be applicable to the updated knowledge in the repository or might transform the knowledge such that the knowledge in the knowledge bases is inconsistent with the knowledge in the repository (i.e., the transformation might not be correctness preserving on the updated knowledge). In a similar vein when a new knowledge base is connected to the knowledge repository, two analogous problems can occur. Either the repository does not contain the information necessary for the new knowledge base or the transformations cannot transform the knowledge into the form that the new knowledge base needs.

Updating the knowledge repository could be done using knowledge acquisition techniques. These can be seen as another type of transformation. These types of transformations must be handled differently because there is a person at one end of the transformation instead of a computer, but the basic structure of transforming one representation (a user language) into the use-neutral representation is the same. For instance a dialogue apparatus can be used to elicit knowledge from a person, but a dialog apparatus is difficult to use in a transformation between two computers. Transformations may also connect the use-neutral representation to people so as to give them direct access to the knowledge in the repository. Several different transformations (in each direction) may be necessary since the language used by different people (i.e., designers, manufacturers, maintainers) may not necessarily be the same. Transformations between people and machines might involve a form (possibly simplified) of natural language understanding and text generation.

Gateway security will also be an issue. Vendors must be allowed access to non-proprietary knowledge while users within Boeing must be allowed greater access. Even within Boeing there might be separate grades of accessibility, a factory shop-floor might not be allowed the same access rights as other areas.

# 4 Survey

In the last few years much work has concerned developing and maintaining large knowledge bases. Very little of this has focused on developing and maintaining large knowledge bases which support multiple uses. There has been some research involved in deriving a "shallow" model from a "deeper" one but not for systems with more than two models. Davis [2] has a system which can derive a sequence of models for use in troubleshooting digital circuits, but it cannot generate different models for different uses.

Rich Keller[1] has demonstrated the multiple use capability of the Stanford modeling project. Keller's [5] system derives two sequences of models, one for diagnosis and one for re-design. This research demonstrated the multiple use capability of the Stanford modeling project. He took approximately

[1] Previously on the Stanford project, now at NASA Ames.

20 diagnosis rules from a 150 rule Lockheed expert system for diagnosis of the Reaction Wheel Assembly (RWA) for the Hubble Space Telescope. Discussions with domain experts derived approximately 5 plausible redesign rules for the RWA. Both of these sets of rules were represented in the expert system shell Strobe. Given these two sets of rules a use-neutral representation was formulated and transformations for connecting these two specific rule sets with the use-neutral representation were derived.

A representation language/s must be chosen for the single source repository. As was stated earlier a single language may not be capable of expressing all the types of knowledge which must be stored in the repository. This language must be capable of representing both deductive and heuristic information and in a form which is independent of its intended use. There are several efforts which deal with creating representation languages which are capable of representing "all" knowledge regardless of use: CYC at MCC [7] and modeling of scientific and engineering device knowledge at Stanford [4]. These representation languages and their associated tools might be suitable for representing the knowledge in our single source repository. The domain representation within these languages must also be determined. It is possible that none of these projects have a representation which will suit our purposes. The Stanford project, since it is also dealing with engineering devices, is more likely to have a predefined problem representation which will suit our purposes. The MCC project is trying to represent "all" knowledge. We are, as is Stanford, trying to solve the easier problem of representing "relevant engineering" knowledge. We have purposefully not spent a lot of time exploring these representation languages. We feel that the possible choice of a representation language should be driven by the needs of the use-neutral representation. We plan to have a first cut of what the requirements of the representation are before we explore these representation languages in depth. This will help to avoid a preconceived bias on our parts.

Research in reformulating knowledge has also been pursued in the last few years. There has been work on abstraction transformations and to a lesser extent on approximation abstractions [6, 3, 14, 15]. Within correctness preserving transformations there has been work on both shifting between expert system shells (i.e., analogous to compilers) [10, 13, 11] and shifting between problem representations (i.e., traditional reformulation) [1, 8, 12, 9]. None of this work has been based on large knowledge bases and therefore not explored certain relevant issues (e.g., knowledge base management, knowledge base maintenance, and version control).

# 5 Focus

Previously we gave an overview of all the issues related to a single source knowledge repository. It is important at this point to emphasis which of these issues this project is emphasizing. These aspects of the system were chosen to allow the system to have a functional value to Boeing within 5 years while research on the other issues continues. The following four issues are the focus of this project.

A representation language is needed which is capable of expressing the information necessary to handle all the different desired uses of the knowledge. This is a tall order. The approach we take is to try and circumscribe the uses to which we will expect this knowledge to be put. The goal is to represent sufficient knowledge to achieve this set of uses as opposed to any use defined later on . This representation language should be flexible enough to allow extensions to the set of pre-defined uses. It is also apparent that one pre-existing representation language may not be suitable for handling all the different type of knowledge necessary (i.e, deductive knowledge versus heuristic knowledge). Portions of multiple pre-existing representation languages may have to be used within the knowledge repository. Bear in mind that this does not necessarily dictate that redundant knowledge will be represented.

Transformations are necessary to down-load the knowledge in the repository to multiple knowledge bases. We plan to explore three basic types of transformations: abstraction, approximation, and correctness preserving. Frequently the full model of a system is too complex to analyze, due to computational restrictions. In these circumstances a simpler model of the system must be used for the analysis. The transformation from the full model to a simpler one is an abstraction. These are used frequently in engineering domains where the complexity of the full system is often overwhelming. Approximations are a similar type of transformation. An abstracted model is correct and must just be refined to arrive at the original full model. An approximate model is actually incorrect to some allowed degree of error (i.e., the approximate model cannot be refined into the original full model). These types of transformations are also used frequently throughout engineering domains (i.e., the simplex method). The above two types of transformations are homomorphisms. Correctness preserving transformations change one model into an equivalent model (i.e., it is an isomorphism). Two examples of when this type of transformation is important is a shift between the problem solvers' associated language (i.e., OPS5 to Prolog) or a problem reformulation. A problem reformulation is a shift from one representation of a problem to another within the same representation language. For instance the shift between a model containing **gender** and **parent** to a model containing **mother** and **father**. It turns out that this last type of transformation is very important in achieving a model of the problem which is computationally efficient for a particular problem solver.

The ability to update the single source knowledge repository such that the transformations to the knowledge bases are still applicable and correctness preserving is very important. Note that a transformation between two representations could trivially be $A \Rightarrow B$. But this type of representation is not very general and therefore will not be applicable if $A$ is updated to $A'$. It is very difficult to create transformations which allow all possible types of updates. We plan to circumscribe the set of updates for which the transformations are guaranteed to still be applicable. This allows the system to actually be used with production level knowledge bases and thus simplify a class of their update problems and lower their maintenance costs while research on broadening the class of guaranteed updates is continued.

The ability to decide later on to connect a new knowledge base to the knowledge repository is also important. Analogous to the problem with updating, it is very difficult to allow the easy connection of any new knowledge base. We plan to circumscribe the set of knowledge bases for which the transformations are guaranteed to connect. This allows the system to actually be used with production level knowledge bases and thus simplify the creation of a class of new knowledge bases and lower their start-up costs while research on broadening the class of connectable knowledge bases is continued.

# 6 Progress

The progress made by this project in slightly under 3 person-months was a feasibility study using a knowledge base for the F-111 hydraulics system. We established a use-neutral knowledge repository for the F-111 hydraulics system and a set of transformations from this knowledge repository to multiple knowledge bases each for a specific task. In this feasibility study there were four knowledge bases derived: one knowledge base for design using a model-based reasoning problem solver[2]; two knowledge bases (at different levels of abstraction) for diagnosis using a model-based reasoning problem solver; and one knowledge base for diagnosis using an associational reasoning problem solver. The derived associational diagnosis rules were more precise than person derived rules. We demonstrated the connection of a pre-specified knowledge base to the use-neutral knowledge repository; in doing this we discovered inconsistencies in the pre-specified knowledge base. We demonstrated the propagation of updates, which were made to the use-neutral knowledge repository, down to the pertinent knowledge bases. We also demonstrated a partially automated methodology for the derivation of a use-neutral knowledge repository and its associated transformations from a set of related use-specific knowledge bases.

## 6.1 F-111 Hydraulic Models

The hydraulic system of the F-111 aircraft is shown in Figure 4. The hydraulic system is broken into

[2] The use of model-based reasoning problem solvers for design is just beginning to be explored[16].

two sub-systems; a primary and a utility subsystem. Within both of these subsystems, there are redundant modules (i.e., left and right modules) and a pressure indicator. Each of these modules consists of a pump and a pressure indicator. The right modules of each subsystem share an engine, as do the left. The system is assumed to be given the flow-demand and throttle settings for each of the engines.

| *Abbreviations* | |
|---|---|
| p | primary |
| pl | primary.left |
| pr | primary.right |
| ip | intermediate.primary |
| ipl | intermediate.primary.left |
| ipr | intermediate.primary.right |
| u | utility |
| ul | utility.left |
| ur | utility.right |
| iu | intermediate.utility |
| iul | intermediate.utility.left |
| iur | intermediate.utility.right |
| le | left.engine |
| re | right.engine |

| *Constants* | |
|---|---|
| Trcf | throttle.rpm.conversion.factor |
| Rtfcc | rpm.to.flow.conversion.constant |
| Ec | Engine constant |
| Ku | utility pipe constant |
| Kp | primary pipe constant |
| Kul | utility left pipe constant |
| Kur | utility right pipe constant |
| Kpl | primary left pipe constant |
| Kpr | primary right pipe constant |

| *Status Variables* | |
|---|---|
| p.pressure.status | u.pressure.status |
| pl.pressure.status | pl.status |
| pr.pressure.status | pr.status |
| ul.pressure.status | ul.status |
| ur.pressure.status | ur.status |
| le.status | re.status |

Figure 3: Abbreviations and Constants

In the models for the knowledge bases and repository, there are some constant values and common abbreviations used in specifying variable names. The models also contain references to boolean status variables that are assumed to indicate the operational status of certain components. These are shown in Figure 3. In each of the task-specific models, we indicate which of the model variables are assumed to be instantiated prior to use of that model. There were four constraints which were common to all the derived models (see Figure 5). The relationships between the models are shown in Figure 6. The use-neutral model has 17 constraints on the components; these are shown in Figure 7. These models are used to illustrate the technology involved with a use-neutral knowledge repository; there is no claim that they precisely reflect the actual physics involved.

In the Diagnosis I model (see Figure 8), the areas of the pipes are fixed and the resistance of the pipe has been abstracted (i.e., the pipe resistance is 0). In the Diagnosis II model (see Figure 9), the areas of the pipes are fixed but the resistance of the pipe is not abstracted. The Design model (see Figure 10) also abstracts the resistance of the pipe but does not fix the areas of the pipes (i.e., while during diagnosis the pipes should not be allowed to change size, this is desirable in a design model).

**Chronology** These models were derived as follows. Diagnosis I was chosen as our initial model; it was previously developed by another research project in the Advanced Technology Center. Using this model, high-level descriptions of the Diagnosis II (by adding the notion of pipe resistance) and Design (by adding the notion that pipe radii can change) models were hypothesized. Using these three models (two of which were only high-level descriptions) and knowledge of basic physics, the use-neutral model was hypothesized. Using the use-neutral model, the assumptions made by the various models, and a set of general transformations; the exact set of constraints specified in Diagnosis I and a plausible set for Diagnosis II and Design were derived.

## 6.2 Transformations

The transformations used in deriving the task-specific models will now be discussed. The reformulation of the use-neutral into Diagnosis I is not shown here; it is a combination of the transformations used to reach the other two models.

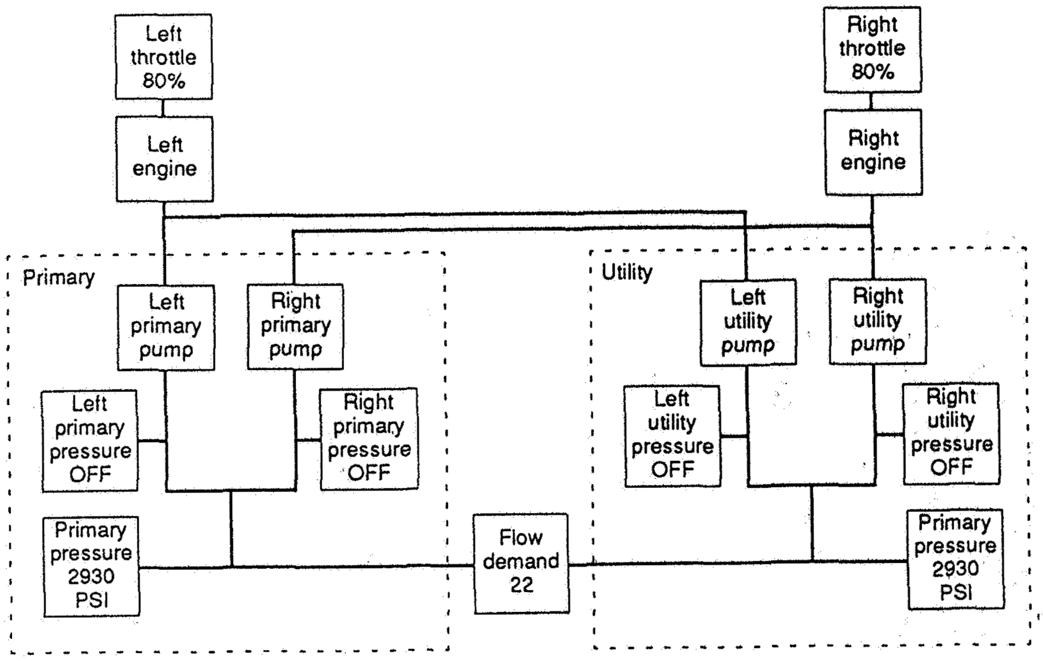


Figure 4: Hydraulic System for F-111

```
1 flow-demand-rule
if T then u.flow.demand = sys.flow.demand/2
if T then p.flow.demand = sys.flow.demand/2

2 flow-large-rule
if T then u.flow = ul.flow+ur.flow
if T then p.flow = pl.flow+pr.flow

16 reading-rule
if u.pressure.status=0 then
   u.pressure.indicator = u.pressure
if p.pressure.status=0 then
   p.pressure.indicator = p.pressure

17 pressure-indicator-constraint
if pl.pressure.status=0 then
   if pl.pressure>1300 then pl.pressure.indicator=1
      else pl.pressure.indicator=0
if pr.pressure.status=0 then
   if pr.pressure>1300 then pr.pressure.indicator=1
      else pr.pressure.indicator=0
if ul.pressure.status=0 then
   if ul.pressure>1300 then ul.pressure.indicator=1
      else ul.pressure.indicator=0
if ur.pressure.status=0 then
   if ur.pressure>1300 then ur.pressure.indicator=1
      else ur.pressure.indicator=0
```

Figure 5: Rules common to ALL Models

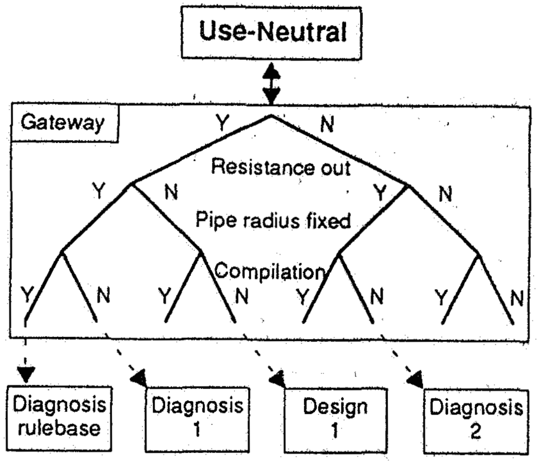


Figure 6: Relationships between Models

**Use-Neutral to Diagnosis II** This is a fairly straight forward set of transformations. First the assumptions are added (i.e., that the radii are constant). This allows the use-neutral constraints (UNC) 4 and 10 in Figure 7 to become constants (i.e., the pipe areas are now constants). The intermediate results computed in UNC 5 can be folded into UNC 3 since no other constraint refers to them and they are not tested by a sensor. This gives the set of constraints shown in Figure 9. This highlights two general transformation techniques: compiling constraints into constants and folding constraints into each other.

| *Inputs* | |
|---|---|
| sys.flow.demand | fluid.density |
| le.throttle | re.throttle |
| u.radius | p.radius |
| ul.radius | pl.radius |
| ur.radius | pr.radius |

*3 flow-small-rule*

```
if ul.status=0 then
  $ul.flow = ul.area \times fluid.density \times ul.velocity$
if ur.status=0 then
  $ur.flow = ur.area \times fluid.density \times ur.velocity$
if pl.status=0 then
  $pl.flow = pl.area \times fluid.density \times pl.velocity$
if pr.status=0 then
  $pr.flow = pr.area \times fluid.density \times pr.velocity$
```

*4 area-small-rule*

```
if T then $ul.area = \pi \times ul.radius^2$
if T then $ur.area = \pi \times ur.radius^2$
if T then $pl.area = \pi \times pl.radius^2$
if T then $pr.area = \pi \times pr.radius^2$
```

*5 velocity-rule*

```
if ul.status=0 then $ul.velocity = Ec \times le.rpm$
if ur.status=0 then $ur.velocity = Ec \times re.rpm$
if pl.status=0 then $pl.velocity = Ec \times le.rpm$
if pr.status=0 then $pr.velocity = Ec \times re.rpm$
```

*6 rpm-rule*

```
if le.status=0 then $le.rpm = Trcf \times le.throttle$
if re.status=0 then $re.rpm = Trcf \times re.throttle$
```

*7 final-pressure-rule*

```
if $u.flow \geq u.flow.demand$ then $u.pressure=2930$
  else $u.pressure=2930 \times (u.flow/u.flow.demand)$
if $p.flow \geq p.flow.demand$ then $p.pressure=2930$
  else $p.pressure=2930 \times (p.flow/p.flow.demand)$
```

*8 intermediate-large-pressure-rule*

```
if ul.status=0 & ur.status=0 then
  $iu.pressure = u.pressure+(u.resistance \times u.flow^2)$
if pl.status=0 & pr.status=0 then
  $ip.pressure = p.pressure+(p.resistance \times p.flow^2)$
```

Figure 7: Use-Neutral Model

*9 resistance-large-rule*

```
if T then $u.resistance = Ku/(2 \times fluid.density \times u.area^2)$
if T then $p.resistance = Kp/(2 \times fluid.density \times p.area^2)$
```

*10 area-large-rule*

```
if T then $u.area = \pi \times u.radius^2$
if T then $p.area = \pi \times p.radius^2$
```

*11 force-large-rule*

```
if ul.status=0 & ur.status=0 then
  $u.force = iu.pressure \times u.area$
if pl.status=0 & pr.status=0 then
  $p.force = ip.pressure \times p.area$
```

*12 force-small-rule*

```
if ul.status=0 & ur.status=0 then
  $u.force = ur.force+ul.force$
if pl.status=0 & pr.status=0 then
  $p.force = pr.force+pl.force$
```

*13 intermediate-small-pressure-rule*

```
if ul.status=0 then $iul.pressure = ul.force/ul.area$
if ur.status=0 then $iur.pressure = ur.force/ur.area$
if pl.status=0 then $ipl.pressure = pl.force/pl.area$
if pr.status=0 then $ipr.pressure = pr.force/pr.area$
```

*14 pressure-small-rule*

```
if ul.status=0 then $ul.pressure =$
  $iul.pressure+(ul.resistance \times ul.flow^2)$
if ur.status=0 then $ur.pressure =$
  $iur.pressure+(ur.resistance \times ur.flow^2)$
if pl.status=0 then $pl.pressure =$
  $ipl.pressure+(pl.resistance \times pl.flow^2)$
if pr.status=0 then $pr.pressure =$
  $ipr.pressure+(pr.resistance \times pr.flow^2)$
```

*15 resistance-small-rule*

```
if T then $ur.resistance=Kur/(2 \times fluid.density \times ur.area^2)$
if T then $ul.resistance=Kul/(2 \times fluid.density \times ul.area^2)$
if T then $pr.resistance=Kpr/(2 \times fluid.density \times pr.area^2)$
if T then $pl.resistance=Kpl/(2 \times fluid.density \times pl.area^2)$
```

Figure 7 Continued: Use-Neutral Model

| *Inputs* | |
|---|---|
| sys.flow.demand | |
| re.throttle | le.throttle |

*3 flow-small-rule*

```
if ul.status=0 then ul.flow = Rtfcc×le.rpm
if ur.status=0 then ur.flow = Rtfcc×re.rpm
if pl.status=0 then pl.flow = Rtfcc×le.rpm
if pr.status=0 then pr.flow = Rtfcc×re.rpm
```

*4 rpm-rule*

```
if le.status=0 then le.rpm = Trcf×le.throttle
if re.status=0 then re.rpm = Trcf×re.throttle
```

*5 final-pressure-rule*

```
if u.flow≥u.flow.demand then u.pressure=2930
  else u.pressure=2930×(u.flow/u.flow.demand)
if p.flow≥p.flow.demand then p.pressure=2930
  else p.pressure=2930×(p.flow/p.flow.demand)
```

*6 pressure-small-rule*

```
if ul.status=0 then
  if ul.flow<u.flow.demand/2 then
    ul.pressure=0 else ul.pressure=u.pressure
if ur.status=0 then
  if ur.flow<u.flow.demand/2 then
    ur.pressure=0 else ur.pressure=u.pressure
if pl.status=0 then
  if pl.flow<p.flow.demand/2 then
    pl.pressure=0 else pl.pressure=p.pressure
if pr.status=0 then
  if pr.flow<p.flow.demand/2 then
    pr.pressure=0 else pr.pressure=p.pressure
```

Figure 8: Diagnosis I Model

| *Inputs* | |
|---|---|
| sys.flow.demand | fluid.density |
| re.throttle | le.throttle |
| u.area | p.area |
| ul.area | ur.area |
| pl.area | pr.area |

*3 flow-small-rule*

```
if ul.status=0 then ul.flow = Rtfcc×le.rpm
if ur.status=0 then ur.flow = Rtfcc×re.rpm
if pl.status=0 then pl.flow = Rtfcc×le.rpm
if pr.status=0 then pr.flow = Rtfcc×re.rpm
```

*4 rpm-rule*

```
if le.status=0 then le.rpm = Trcf×le.throttle
if re.status=0 then re.rpm = Trcf×re.throttle
```

*5 final-pressure-rule*

```
if u.flow≥u.flow.demand then u.pressure=2930
  else u.pressure=2930×(u.flow/u.flow.demand)
if p.flow≥p.flow.demand then p.pressure=2930
  else p.pressure=2930×(p.flow/p.flow.demand)
```

*6 intermediate-large-pressure-rule*

```
if ul.status=0 & ur.status=0 then
  $iu.pressure=u.pressure+(u.resistance\times u.flow^2)$
if pl.status=0 & pr.status=0 then
  $ip.pressure=p.pressure+(p.resistance\times p.flow^2)$
```

*7 resistance-large-rule*

```
if T then $u.resistance=Ku/(2\times fluid.density\times u.area^2)$
if T then $p.resistance=Kp/(2\times fluid.density\times p.area^2)$
```

*8 force-large-rule*

```
if ul.status=0 & ur.status=0 then
  iu.pressure = u.force/u.area
if pl.status=0 & pr.status=0 then
  ip.pressure = p.force/p.area
```

*9 force-small-rule*

```
if ul.status=0 & ur.status=0 then
  u.force = ur.force+ul.force
if pl.status=0 & pr.status=0 then
  p.force = pr.force+pl.force
```

Figure 9: Diagnosis II Model

*10 intermediate-small-pressure-rule*

```
if ur.status=0 then iur.pressure = ur.force/ur.area
if ul.status=0 then iul.pressure = ul.force/ul.area
if pr.status=0 then ipr.pressure = pr.force/pr.area
if pl.status=0 then ipl.pressure = pl.force/pl.area
```

*11 pressure-small-rule*

```
if ul.status=0 then ul.pressure =
  iul.pressure+(ul.resistance×ul.flow$^2$)
if ur.status=0 then ur.pressure =
  iur.pressure+(ur.resistance×ur.flow$^2$)
if pl.status=0 then pl.pressure =
  ipl.pressure+(pl.resistance×pl.flow$^2$)
if pr.status=0 then pr.pressure =
  ipr.pressure+(pr.resistance×pr.flow$^2$)
```

*12 resistance-small-rule*

```
if T then ur.resistance=Kur/(2×fluid.density×ur.area$^2$)
if T then ul.resistance=Kul/(2×fluid.density×ul.area$^2$)
if T then pr.resistance=Kpr/(2×fluid.density×pr.area$^2$)
if T then pl.resistance=Kpl/(2×fluid.density×pl.area$^2$)
```

Figure 9 Continued: Diagnosis II Model

**Use-Neutral to Design** This is a more complex set of transformations; the intermediate constraints are shown in Figure 11. First the assumptions are added (i.e., that the pipe resistances are 0). This allows UNC 9 and 15 to become constants; these constants are replaced in UNC 8 and 14 to produce 8′ and 14′. Since 8′ and 14′ are now equalities, they can be substituted into UNC 11 and 13 and removed. This produces 11′ and 13′. Notice that this can only be done since the antecedents of constraints 8′ and 14′ are implied by the antecedents of constraints UNC 11 and 13 (this type of requirement occurs throughout these transformation sequences).

The assumptions that the left and right pipes for each subsystem have equal sizes and that the pipes out of each subsystem are twice the size of these inflow pipes are added; transforming 13′ into 13″. The constraint 11′ is now substituted into the constraint UNC 12 and removed; producing 12′. The constraint 12′ is substituted into the constraint 13″ and removed; producing 13‴. At this point UNC 10 can also be removed[2]. The assumption that the forces through the left and right modules of each subsystem are equal is added. This allows the in-

| *Inputs* | |
|---|---|
| sys.flow.demand | fluid.density |
| re.throttle | le.throttle |
| ul.radius | ur.radius |
| pl.radius | pr.radius |

*3 flow-small-rule*

```
if ul.status=0 then
  ul.flow = ul.area×fluid.density×ul.velocity
if ur.status=0 then
  ur.flow = ur.area×fluid.density×ur.velocity
if pl.status=0 then
  pl.flow = pl.area×fluid.density×pl.velocity
if pr.status=0 then
  pr.flow = pr.area×fluid.density×pr.velocity
```

*4 area-small-rule*

```
if T then ul.area = $\pi$×ul.radius$^2$
if T then ur.area = $\pi$×ur.radius$^2$
if T then pl.area = $\pi$×pl.radius$^2$
if T then pr.area = $\pi$×pr.radius$^2$
```

*5 velocity rule (small pipe)*

```
if T then ul.velocity = Ec×le.rpm
if T then ur.velocity = Ec×re.rpm
if T then pl.velocity = Ec×le.rpm
if T then pr.velocity = Ec×re.rpm
```

*6 rpm-rule*

```
if le.status=0 then le.rpm = Trcf×le.throttle
if re.status=0 then re.rpm = Trcf×re.throttle
```

*7 final-pressure-rule*

```
if u.flow≥u.flow.demand then u.pressure=2930
  else u.pressure=2930×(u.flow/u.flow.demand)
if p.flow≥p.flow.demand then p.pressure=2930
  else p.pressure=2930×(p.flow/p.flow.demand)
```

*8 pressure-small-rule*

```
if ul.status=0 then
  if ul.flow<u.flow.demand/2 then
    ul.pressure=0 else ul.pressure=u.pressure
if ur.status=0 then
  if ur.flow<u.flow.demand/2 then
    ur.pressure=0 else ur.pressure=u.pressure
if pl.status=0 then
  if pl.flow<p.flow.demand/2 then
    pl.pressure=0 else pl.pressure=p.pressure
if pr.status=0 then
  if pr.flow<p.flow.demand/2 then
    pr.pressure=0 else pr.pressure=p.pressure
```

Figure 10: Design Model

```
8' intermediate-large-pressure-rule
if ul.status=0 & ur.status=0 then
  iu.pressure = u.pressure
if pl.status=0 & pr.status=0 then
  ip.pressure = p.pressure

14' pressure-small-rule
if ul.status=0 then ul.pressure = iul.pressure
if ur.status=0 then ur.pressure = iur.pressure
if pl.status=0 then pl.pressure = ipl.pressure
if pr.status=0 then pr.pressure = ipr.pressure

11' force-large-rule
if ul.status=0 & ur.status=0 then
  u.force = u.pressure×u.area
if pl.status=0 & pr.status=0 then
  p.force = p.pressure×p.area

13' intermediate-small-pressure-rule
if ul.status=0 then ul.pressure = ul.force/ul.area
if ur.status=0 then ur.pressure = ur.force/ur.area
if pl.status=0 then pl.pressure = pl.force/pl.area
if pr.status=0 then pr.pressure = pr.force/pr.area

13'' intermediate-small-pressure-rule
if ul.status=0 then ul.pressure = (2*ul.force)/u.area
if ur.status=0 then ur.pressure = (2*ur.force)/u.area
if pl.status=0 then pl.pressure = (2*pl.force)/p.area
if pr.status=0 then pr.pressure = (2*pr.force)/p.area

12' force-small-rule
if ul.status=0 & ur.status=0 then
  u.pressure×u.area = ur.force+ul.force
if pl.status=0 & pr.status=0 then
  p.pressure×p.area = pr.force+pl.force

13''' intermediate-small-pressure-rule
if ul.status=0 then ul.pressure =
  (2*ul.force)/((ur.force+ul.force)/u.pressure)
if ur.status=0 then ur.pressure =
  (2*ur.force)/((ur.force+ul.force)/u.pressure)
if pl.status=0 then pl.pressure =
  (2*pl.force)/((pr.force+pl.force)/p.pressure)
if pr.status=0 then pr.pressure =
  (2*pr.force)/((pr.force+pl.force)/p.pressure)

13'''' intermediate-small-pressure-rule
if ul.status=0 then ul.pressure = u.pressure
if ur.status=0 then ur.pressure = u.pressure
if pl.status=0 then pl.pressure = p.pressure
if pr.status=0 then pr.pressure = p.pressure
```

Figure 11: Intermediate Constraints

termediate values computed in $12'$ to be removed[3] and $13'''$ is transformed into $13''''$. The last two assumptions added are that the flows for each module in a subsystem are equal and that the pressure for each module is either 0 or its maximum value (i.e., 2930) which alters $13''''$ into constraint 8 of the design model (see Figure 10). It is important to note that for the task-specific knowledge base to be consistent some of the assumptions have to explicitly remain (i.e., they cannot be totally compiled into the previous constraints). These assumptions are not explicitly shown in our models. For instance for the Design model the assumptions which are not totally compiled are $\mathit{ul.area}=\mathit{ur.area}$, $\mathit{pl.area}=\mathit{pr.area}$, $\mathit{ul.force}=\mathit{ur.force}$, $\mathit{pl.force}=\mathit{pr.force}$, $\mathit{ul.pressure} = 2930 \vee \mathit{ul.pressure} = 0$, $\mathit{ur.pressure} = 2930 \vee \mathit{ur.pressure} = 0$, $\mathit{pl.pressure} = 2930 \vee \mathit{pl.pressure} = 0$, $\mathit{pr.pressure} = 2930 \vee \mathit{pr.pressure} = 0$.

**Consistency versus Pre-defined KB** If a pre-defined knowledge base is added to the system a choice must frequently be made. Either the transformations are applied consistently given a specific set of assumptions or they are partially applied so as to retain the exact representation of the original knowledge base. When the latter is chosen, the inconsistent application of the transformations can allow the knowledge base to be inconsistent. The inconsistent application of transformations can be a flag highlighting these inconsistencies and bringing them to the attention of the creator of the knowledge base. For instance in the Diagnosis I model shown in Figure 8, constraints 5 and 6 are inconsistent with each other. This occurs because the assumption that the pressure for each module is either 0 or the maximum is compiled into constraint 6 but not into constraint 5. This inconsistency was preserved so as to achieve the exact representation of the original pre-defined set of constraints for Diagnosis I. The use of our technique highlighted this inconsistency which was not noticed before. Even if the transformations are applied consistently, the

[3] Since the force variables are not input or appear in any other formulas, this formula can only be used to determine values for these forces. And since these forces are never sensed, there is no need to retain the constraints.

derived knowledge base is not necessarily consistent with the original one. But it is consistent modulo its initial input (i.e., the use-neutral knowledge repository, assumptions made, and transformations used).

## 6.3 Associational Rules

An algorithm for generating an associational knowledge base for diagnosis was demonstrated. Figure 12 shows a rule for the Diagnosis I model. The associational rules were derived using a back-propagation algorithm over the results of a model-based reasoner. For instance the previous rule was derived when the symptom *ur.pressure.status = 1* & *ur.pressure.status = 0* and fault *ur.pressure.indicator* were discovered by a model-based reasoner. The symptom was back-propagated over the constraints associated with each component. When the result is totally in terms of the primitives of the model-base reasoner, then the back-propagation terminates and the resulting expression is the antecedent of the associational rule. The rule in Figure 12 was more precise than the rule derived for the same scenario by the person who created the model-base reasoning model.

```
if ur.pressure.status.reading=1&ur.pressure.status=0&
   le.status=0&re.status=0&ul.status=0&ur.status=0&
   [(Rtfcc×Trcf×(le.throttle+re.throttle))>
      (0.22×sys.flow.demand)]&
    (Rtfcc×Trcf×re.throttle)>=.25×sys.flow.demand
 then faulty(ur.pressure.indicator)
```

Figure 12: Associational Diagnosis Rule

## 6.4 Updates

This methodology is most beneficial when updates are performed on the use-neutral knowledge repository and then propagated to the pertinent knowledge bases. We explored the propagation of updates by running the entire updated knowledge repository through the same transformations again to derive the changes to the knowledge bases. Two specific updates were explored in depth. The first adds the notion of *work* to the knowledge repository in terms of its relationship to force. The transformations determine that this concept is irrelevant for these tasks and derive the same set of knowledge bases as before. Then a second update removes force. After this update the transformations produce knowledge bases which use work divided by distance as a replacement for force in all the pertinent constraints.

There are several problems with this method of updating. The transformations should be applied incrementally after an update. Instead of reprocessing the entire knowledge repository, only the pertinent subportions are reprocessed. This is very difficult, because an update may cause another piece of knowledge in the repository to be transformed differently. Also transformations might be applicable where they were not applicable before the update. Even more difficult is the requirement that no other transformations (outside the set which were applied before the update) are applicable now. Further a change to the repository might make the old transformations invalid. When this happens the knowledge repository might no longer be able to reach the existing representation of the knowledge base anymore.

Handling updates is a hard problem, but classes of updates can be defined which are guaranteed to transform correctly. We must explore updates in many domains so as to define this class.

## 6.5 Creating a Knowledge Repository

The research done in this area was very preliminary in nature. Given several pre-existing knowledge bases which contain overlapping knowledge, the generation of a knowledge repository can be partially automated. Their knowledge is compared and the most primitive components of the knowledge bases are used to create the knowledge repository. This can be done via partial matching as follows. Given two knowledge bases, all those rules which are an exact match are automatically placed in the use-neutral repository. For pairs of rules which score high on the partial match, assumptions are hypothesized which allow one of the rules to be transformed into the other. At this point user intervention could avoid nonplausible assumptions. The use-neutral repository is constructed from the knowledge base rules to which assumptions were added to reach other rules. The complexity of this technique increases with the number of knowledge bases, but the number of high scoring partial matches increase inversely.

Notice that this technique relies on one rule being an abstraction of another. This methodology will not work when none of the knowledge bases have the most primitive knowledge, but instead are all slightly different abstractions of this knowledge. For instance, assume the primitive knowledge is $A{=}B{\times}C^2$. If one knowledge base had $A{=}0$ and the other knowledge base had $A{=}100{\times}B$, then the system will not be able to hypothesis the primitive knowledge on its own. It assumes that $A{=}B{\times}C$ was the primitive knowledge using the assumptions that $C{=}0$ or $C{=}100$. There is no reason for it to choose the more complex (and in this case correct) set of assumptions. Either a person must aid the derivation of the knowledge repository or another knowledge source containing general purpose primitive knowledge (e.g., a fluid dynamics knowledge seed) must be brought to bear. If the knowledge bases representations are too different the system also fails. For instance comparing the notion of *equalitaral-triangle* and *triangle-where-all-the-sides-are-equal-length.*

We must explore more domains, to determine what techniques will be useful in which situations.

# 7 Future Research

Exploration of this technology's capacity for generating new knowledge-based systems is needed. First cuts at totally new knowledge bases can be generated. For instance assume that a design knowledge base for a device A and a diagnosis knowledge base for a device B are presently connected to the knowledge repository, we need to derive a diagnostic knowledge base for device A. The same knowledge about the device used in deriving the design knowledge base should be used, but it should now be transformed for a diagnostic task. To derive a first cut at this knowledge base the knowledge concerning device A can be run through the set of transformations previously used on device B. Some transformations are likely not to fire and other necessary transformations are likely to be missing, but the hypothesis is that a good first cut at a knowledge base is generated.

The next logical step in this research is to implement these ideas so as to test them in several domains. To explore the update question in depth, a more complex domain is necessary. We are looking for a NASA domain which meets these requirements. More work is needed on the initial derivation of the knowledge repository and on transformations between multiple problem solvers. Transformations based on approximations (as well as abstractions) and the handling of inverse updates (i.e., updating a knowledge base which is then propagated up to the knowledge repository and then back down to the other pertinent knowledge bases) should be explored.

# 8 Summary

In the life cycle of complex devices (i.e., the docking bay door of the space station) , there are many processes (e.g., design, process planning, and diagnosis) that can be partially automated by knowledge-based systems. As these knowledge-based systems proliferate, the overlap of knowledge amongst these systems leads to increased knowledge management costs (e.g., consistency and configuration management). A knowledge repository capable of feeding multiple knowledge-based systems is one solution. However, it is critical that knowledge in the repository be stored in a use-neutral format and then transformed into representations that are tailored for specific uses (e.g., design). This explicit decomposition of knowledge into a use-neutral corpus, knowledge transformations on that corpus, and use-specific knowledge that is not shared by multiple systems, is useful not only in maintaining existing knowledge-based systems, but also in generating initial versions of entirely new systems.

We have developed a use-neutral representation of an hydraulic system for the F-111 airplane. We demonstrated the ability to derive portions of four different knowledge bases from this use-neutral representation: one knowledge base is for re-design of the device using a model-based reasoning problem solver; two knowledge bases, at different levels of abstraction, are for diagnosis using a model-based reasoning problem solver; and one knowledge base is for diagnosis using an associational reasoning problem solver. The use of our technique highlighted inconsistencies which were not noticed before. The associational rules derived were more precise than the rules derived for the same scenarios by the person who created the model-base reasoning model. We have shown how updates issued against the single source use-neutral knowledge repository can be propagated to the underlying knowledge bases. We have also shown a plausible methodology for generating the knowledge repository given a set of pre-existing knowledge bases.

## References

[1] J. Van Baalen. Automated Design of Specialized Representations. In *Proceedings of the Change of Representation and Problem Reformulation Workshop, Price Waterhouse*, 1990.

[2] R. Davis. Diagnostic Reasoning Based on Structure and Behavior. *Artificial Intelligence*, 24, 1984.

[3] N.S. Flann. Learning Appropriate Abstractions for Planning in Formation Problems. In *Proceedings of the Sixth International Conference on Machine Learning, Ithaca, New York*, pages 235–239. Morgan Kaufmann, 1989.

[4] T. Gruber and U. Iwasaki. How things work: Knowledge-based modeling of physical devices.

[5] R.M. Keller. Model Compilation: An Approach to Automated Model Derivation. In *Proceedings of the Change of Representation and Problem Reformulation Workshop, Price Waterhouse*, 1990.

[6] C.A. Knoblock. Learning Problem-Specific Abstraction Hiearchies. In *Proceedings of the Change of Representation and Problem Reformulation Workshop, Price Waterhouse*, 1990.

[7] D. Lenat and R.V. Guha. The world according to cyc. Technical Report ACA-AI-300-88, MCC, 1988.

[8] M.R. Lowry. Category Theory and Homomorphic Abstraction. In *Proceedings of the Change of Representation and Problem Reformulation Workshop, Price Waterhouse*, 1990.

[9] P.J. Riddle. Automating Shifts of Representation. In P. Benjamin, editor, *Change of Representation and Inductive Bias.* Kluwer, 1989.

[10] P. Rothman. Knowledge Transformation. *AI Expert*, 1988.

[11] R.A. Stachowitz and J.B. Combs. Validation of Expert Systems. In *Proceedings of the Twentieth Hawaii International Conference on System Sciences, Kona, Hawaii*, January 1987.

[12] D. Subramanian. A Theory of Justified Reformulations. In *Proceedings of the Change of Representation and Problem Reformulation Workshop, Price Waterhouse*, 1990.

[13] B. Thuraisingham. From Rules to Frames and Frames to Rules. *AI Expert*, 1989.

[14] A. Unruh and P.S. Rosenbloom. Abstraction in Problem Solving and Learning. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, Detroit, Michigan*, pages 681–687. IJCAI, Morgan Kaufmann, August 1989.

[15] D.S. Weld. Discrepancy Driven Selection of Approximation Reformulations. In *Proceedings of the Change of Representation and Problem Reformulation Workshop, Price Waterhouse*, 1990.

[16] B.C. Williams. Interaction-based invention: Designing novel devices from first principles. In *Proceedings of the Eighth National COnfernce on Artificial Intelligence*, 1990.

☆U.S. GOVERNMENT PRINTING OFFICE: 1991- 527- 06#26017

National Aeronautics and
Space Administration

# Report Documentation Page

| | | |
|---|---|---|
| 1. Report No.<br>NASA CP-3110 | 2. Government Accession No. | 3. Recipient's Catalog No. |

| | |
|---|---|
| 4. Title and Subtitle<br>1991 Goddard Conference on Space Applications of Artificial Intelligence | 5. Report Date<br>May 1991 |
| | 6. Performing Organization Code<br>Code 500 |
| 7. Author(s)<br>James L. Rash, Editor | 8. Performing Organization Report No.<br>91B00064 |
| | 10. Work Unit No. |
| 9. Performing Organization Name and Address<br>Mission Operations and Data Systems Directorate<br>NASA/Goddard Space Flight Center, Code 500<br>Greenbelt, Maryland 20771 | 11. Contract or Grant No. |
| | 13. Type of Report and Period Covered<br>Conference Publication |
| 12. Sponsoring Agency Name and Address<br>National Aeronautics and Space Administration<br>Washington, DC 20546 | 14. Sponsoring Agency Code<br>Code 500 |

15. Supplementary Notes

16. Abstract

This publication comprises the papers presented at the 1991 Goddard Conference on Space Applications of Artificial Intelligence held at the NASA/Goddard Space Flight Center, Greenbelt, Maryland, on May 13-15, 1991. The purpose of this annual conference is to provide a forum in which current research and development directed at space applications of artificial intelligence can be presented and discussed. The papers in this proceedings fall into the following areas: Planning and Scheduling Fault Monitoring/Diagnosis/Recovery, Machine Vision, Robotics, System Development, Information Management, Knowledge Acquisition and Representation, Distributed Systems, Tools, Neural Networks, and Miscellaneous Applications.

| | |
|---|---|
| 17. Key Words (Suggested by Author(s))<br>Artificial Intelligence, expert systems, planning, scheduling, fault isolation, fault diagnosis, machine vision, robotics, control, knowledge representation, knowledge acquisition, neural networks, distributed systems | 18. Distribution Statement<br>Unclassified - Unlimited |

| | | | |
|---|---|---|---|
| 19. Security Classif. (of this report)<br>Unclassified | 20. Security Classif. (of this page)<br>Unclassified | 21. No. of pages<br>376 | 22. Price<br>A17 |